#### **9.4 Символы и кодировки**

### Что вы узнаете

Привет! В этом материале вы познакомитесь с основными методами работы с символами, изучите, как добавлять, изменять и удалять символы в строке и узнаете, какие бывают кодировки символов, как конвертировать текст из одной кодировки в другую.

### Перебор символов в строке

Как вам уже известно, строки — это последовательности символов, и часто при решении задач нужно работать с конкретными символами строк. Символы можно заменять, удалять или создавать новые. Для этого в Java есть неплохие инструменты.

Попробуем сделать самое простое — выведем строку в консоль побуквенно. Для этого нужны два действия: представить строку как массив символов и затем применить метод вывода в консоль.

public static void main(String[] args) {  
   String hello = "Hello";   
   char[] chars = hello.toCharArray(); *//получаем массив*  
*for (char c : chars) {*  
*System.out.println(c); //выводим в консоль*  
*}*  
*}*

Метод ****toCharArray()**** возвращает массив символов типа char. Далее мы можем обратиться к любому символу методом перебора.

Запустите этот код и посмотрите на результат. Попробуйте вывести буквы в обратном порядке.

Если вам известно, на какой позиции находится нужный символ, то на помощь придёт другой метод — ****charAt()****. Ему нужно передать номер позиции.

String hello = "Hello";  
char firstChar = hello.charAt(0);  
System.out.println(“Первая буква:” + firstChar);

Результат:

Первая буква: Н

Обратите внимание, что, как и в любом массиве, индексы в строке начинаются с нуля. Попытка передать значение, которое больше, чем длина строки, приведёт к ошибке. Например, такой код:

String hello = "Hello";  
*//передаём значение 5, которое больше максимального индекса*  
*char firstChar = hello.charAt(hello.length());*  
*System.out.println(firstChar);*

вернёт результат:

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: Index 5 out of bounds for length 5  
    at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:55)  
    at java.base/jdk.internal.util.Preconditions$1.apply(Preconditions.java:52)

Посмотрим на другой пример работы с символами строки. Например, вы разрабатываете текстовый редактор, и вам нужна функция подсчёта количества пробелов в тексте. Есть строка с текстом и некоторым количеством пробелов внутри. Чтобы посчитать количество пробелов, нужно пройтись циклом по всем символам в строке и проверить, является ли каждый из этих символов пробелом. Для этого создаём переменную spaceCount (количество пробелов), изначально равную нулю. Далее пишем цикл. Теперь проверяем, является ли каждый символ пробелом. Если символ «равен» пробелу, то мы увеличиваем значение переменной spaceCount на единицу.

String str = "Это тестовая строка для подсчёта пробелов";  
int spaceCount = 0;  
for (int i = 0; i < str.length(); i++) {  
   if (str.charAt(i) == ' ') {  
       spaceCount++;  
   }  
}  
System.out.println("Количество пробелов: " + spaceCount);

Результат:

Количество пробелов: 5

### Работа с пробелами по краям строки

Зачастую приходится работать со строками, которые содержат пробелы в конце. Например, при наборе мы можем ошибиться и нажать пробел после точки. Вот так:

String text = "Some spaces at the end   ";

Теперь, если попытаемся подсчитать количество символов в строке, то результат будет неверным, так как мы можем не знать об этих пробелах.

Для избавления от ненужных символов можно использовать метод ****trim()****.

public static void main(String[] args) {  
   String text = "  Some spaces at the start and at the end   ";  
   System.out.println("Исходный текст " + text);  
   String clearedText = text.trim();  
   System.out.println("Полученный текст " + clearedText);  
}

Результат:

Исходный текст   Some spaces at the start and at the end

Полученный текст Some spaces at the start and at the end

Начиная с 11-й версии, в Java появился новый метод ****strip()****, который убирает не только пробелы, но и другие символы, которые могут считаться пробелами в различных кодировках. Поэтому, если у вас JDK 11-й версии и выше, то предпочтительно использовать метод ****strip()****.

### Методы isDigit и isLetter

Предположим, что у вас есть небольшой текст, и вам нужно проверить, начинается ли он с цифры. Как это сделать? На помощь придёт метод isDigit класса Character — он как раз проверяет, является ли переданный ему в качестве параметра символ цифрой:

public static void main(String[] args) {  
    String text = "1990 year";  
    if (Character.isDigit(text.charAt(0))) {  
        System.out.println("This is a digit");  
    } else {  
            System.out.println("This is not a digit");  
    }  
}

Также есть другие полезные методы, которыми можно пользоваться при написании программ:

****isLetter**** — проверяет, является ли переданный символ буквой;

****isIdeographic**** — проверяет, является ли переданный символ иероглифом;

****isJavaLetterOrDigit**** — проверяет, является ли переданный символ допустимым при именовании переменных и методов в Java.

### Символы и их коды

Вы помните, что информация в компьютере хранится в виде битов, нулей и единиц. Более крупной единицей хранения информации является байт, состоящий из 8 бит — по сути, это число от −128 до 127. Если мы хотим хранить в памяти строки, состоящие из символов, то должны установить такое соответствие этих чисел и конкретных символов. Но всегда помните, что символы можно задавать числами и преобразовывать числа в символы и обратно, а определённым числам соответствуют определённые символы.

Как вам известно из темы с примитивами, каждому символу соответствует определённый числовой код. Например, латинскому символу A в кодировке UTF-8 соответствует код 65.

Ещё раз: ****кодировкой**** (от англ. encoding) называется перечень соответствий конкретных символов определённым числам. Такие числа называют кодами символов, а набор символов в кодировке называется charset — в переводе с английского — «набор символов».

Зачем знать, что такое кодировки и как с ними работать? Существует много кодировок, и вы можете столкнуться с ними на практике. Java — кросс-платформенный язык, и зачастую приходится запускать в системе Windows то, что написано в Linux и наоборот. Исторически эти операционные системы работают с различными кодировками.

### Какие бывают кодировки

Давайте рассмотрим типы кодировок.

1. Кодировки бывают ****однобайтные**** (от англ. single-byte) — это кодировки, в которых один символ представлен одним байтом. Примеры таких кодировок: ASCII; ISO-8856-1; Windows-1251, которая присутствует в русскоязычной версии операционной системы Windows. В такой кодировке можно закодировать 256 чисел и, соответственно, символов. Например, в кодировке ASCII используется только 7 бит, и в ней всего 128 символов: только цифры и знаки препинания, специальные символы и латинские буквы. А вот в кодировке Windows-1251 используются все 8 бит, и там 256 символов, в том числе кириллические буквы.
2. Кодировки бывают ****многобайтные**** (от англ. multibyte). Они позволяют кодировать символы с двумя и большим количеством байт, что даёт возможность закодировать все существующие в мире символы. Вариантов многобайтных кодировок существует много, но однажды разработчики решили создать единый стандарт — Unicode.
3. ****Unicode**** — набор кодировок, в которых пересекающиеся символы (это символы, которые встречаются в нескольких кодировках) имеют одинаковые номера. Примеры кодировок: UTF-8, UTF-16, UTF-32.

****UTF-8****— самая популярная из них. В ней используется переменное число байт — от 1 до 4, в зависимости от того, какой символ они кодируют. Например, латинская буква S кодируется числом 53 — в шестнадцатеричной системе это число легко влезает в один байт. А буква грузинского алфавита весит целых 3 байта.

### Кодировки в Java

Теперь поговорим о Java. Как было сказано выше, в зависимости от операционной системы, на которой установлена Java, кодировка может отличаться. В Java за кодировки отвечает класс java.nio.charset.Charset. С его помощью мы можем узнать, какая кодировка установлена по умолчанию в нашей Java-машине.

System.out.println(Charset.defaultCharset());

Результат:

UTF-8

Также класс Charset используется для задания кодировки текста. Чтобы узнать коды символов в данной кодировке, используем метод getBytes класса String:

public static void main(String[] args) {  
   String text = "Привет";  
   byte[] textBytes = text.getBytes(StandardCharsets.UTF\_8);   
   for (byte textByte : textBytes) {  
      System.out.print(textByte + " ");  
   }  
}

Результат:

-48 -97 -47 -128 -48 -72 -48 -78 -48 -75 -47 -126

Как видите, здесь указываем кодировку, чтобы программа могла применить нужную конвертацию. В другой кодировке числа могут быть другими (но необязательно). Попробуем вывести побайтово символы в кодировке Windows.

public static void main(String[] args) {  
   try {  
   String text = "Привет";  
   byte[] textBytes = text.getBytes("Windows-1251");  
       for (byte textByte : textBytes) {  
           System.out.print(textByte + " ");  
       }  
        } catch (UnsupportedEncodingException e) {  
       throw new RuntimeException(e);  
    }  
}

Результат:

-49 -16 -24 -30 -27 -14

Как видите, здесь кодировка указана строкой (Windows-1251), а не ссылкой (StandardCharsets.UTF-8). Класс StandardCharsets содержит ограниченный список кодировок, и поэтому пришлось искать кодировку по имени. Слова try и catch, которые вы видите, — это специальная конструкция, которая защищает строку кода от возможной ошибки в написании имени кодировки. С исключениями мы познакомимся в одном из следующих модулей.

### Как сменить кодировку текста

Давайте попробуем сменить кодировку заданного текста. В Java это можно сделать с помощью особого конструктора класса String, которому нужно передать в качестве параметров текст в виде набора байтов и названия кодировки.

public static void main(String[] args) {  
   String text = "Hello всем";  
   String encoded = new String(text.getBytes(), StandardCharsets.US\_ASCII); *//создаём новую строку*  
*System.out.println(encoded);*  
*}*

Результат:

Hello ��������

Как вы помните, в кодировке US-ASCII всего 128 символов, поэтому кириллические символы в кодировку US-ASCII не перекодировались.

### Заключение

Итак, в этом материале вы познакомились с основными методами работы с символами строки, научились выводить строки посимвольно, а также искать нужный символ в строке. Узнали, какие бывают кодировки символов и как применять их в Java.