Работа со строками

Символы

Для записи одиночных символов используются следующие формы.

- Печатные символы можно записать в апострофах: ' a ', ' N ', ' ? '.
- Управляющие символы записываются в апострофах с обратной наклонной чертой:
- ' \n ' — символ перевода строки newline с кодом ASCII 10;
- ' \r ' — символ возврата каретки CR с кодом 13;
- ' \f ' — символ перевода страницы FF с кодом 12;
- ' \b ' — символ возврата на шаг BS с кодом 8;
- ' \t ' — символ горизонтальной табуляции HT с кодом 9;
- ' \\ ' — обратная наклонная черта;
- ' \" ' — кавычка;
- ' \' ' — апостроф.
- Код любого символа с десятичной кодировкой от 0 до 255 можно задать, записав его не более чем тремя цифрами в восьмеричной системе счисления в апострофах после обратной наклонной черты: ' \123 ' — буква S , ' \346 ' — буква Ж в кодировке CP1251. Не рекомендуется использовать эту форму записи для печатных и управляющих символов, перечисленных в предыдущем пункте, поскольку компилятор сразу же переведет восьмеричную запись в указанную выше форму. Наибольший код ' \377 ' — десятичное число 255.
- Код любого символа в кодировке Unicode набирается в апострофах после обратной наклонной черты и латинской буквы ц ровно четырьмя шестнад-цатеричными цифрами: ' \u0053 ' — буква S , ' \u0416 ' — буква Ж .

Символы хранятся в формате типа char (см. ниже).

Примечание

Прописные русские буквы в кодировке Unicode занимают диапазон от ' \u0410 ' — заглавная буква А , до ' \u042F ' — заглавная Я , строчные буквы от ' \u0430 ' — а , до ' \044F ' — я .

В какой бы форме ни записывались символы, компилятор переводит их в Unicode, включая и исходный текст программы.

Замечание

Компилятор и исполняющая система Java работают только с кодировкой Unicode.

Строки

Строки символов заключаются в кавычки. Управляющие символы и коды записываются в строках точно так же, с обратной наклонной чертой, но, разумеется, без апострофов, и оказывают то же действие. Строки могут располагаться только на одной строке исходного кода, нельзя открывающую кавычку поставить на одной строке, а закрывающую — на следующей.

Вот некоторые примеры:

```
"Это строка\nс переносом"
"\"Спартак\" — Чемпион!"
```

Замечание

Строки символов нельзя начинать на одной строке исходного кода, а заканчивать на другой.

Для строковых констант определена операция сцеплений, обозначаемая плюсом.

`" Сцепление " + "строк"` дает в результате строку "Сцепление строк" .

Чтобы записать длинную строку в виде одной строковой константы, надо после закрывающей кавычки на первой и следующих строках поставить плюс +; тогда компилятор соберет две (или более) строки в одну строковую константу, например:

```
"Одна строковая константа, записанная "+
"на двух строках исходного текста"
```

Тот, кто попытается выводить символы в кодировке Unicode, например, слово "Россия":

```
System.out.println("\u0429\u043e\u0441\u0441\u0438\u044f");
```

должен знать, что Windows 95/98/ME вообще не работает с Unicode, а Windows NT/2000 использует для вывода в окно **Command Prompt** шрифт Terminal, в котором русские буквы, расположены в начальных кодах Unicode, почему-то в кодировке CP866, и разбросаны по другим сегментам Unicode.

Не все шрифты Unicode содержат начертания (glyphs) всех символов, поэтому будьте осторожны при выводе строк в кодировке Unicode.

Совет

Используйте Unicode напрямую только в крайних случаях.

Очень большое место в обработке информации занимает работа с текстами. Как и многое другое, текстовые строки в языке Java являются объектами. Они представляются экземплярами класса string или класса stringBuffer .

Поначалу это необычно и кажется слишком громоздким, но, привыкнув, вы оцените удобство работы с,классами, а не с массивами символов.

Конечно, возможно занести текст в массив символов типа char или даже в массив байтов типа byte , но тогда вы не сможете использовать готовые методы работы с текстовыми строками.

Зачем в язык введены два класса для хранения строк? В объектах класса string хранятся строки-константы неизменной длины и содержания, так сказать, отлитые в бронзе. Это значительно ускоряет обработку строк и позволяет экономить память, разделяя строку между объектами, использующими ее. Длину строк, хранящихся в объектах класса stringBuffer , можно менять, вставляя и добавляя строки и символы, удаляя подстроки или сцепляя несколько строк в одну строку. Во многих случаях, когда надо изменить длину строки типа string , компилятор Java неявно преобразует ее к типу stringBuffer , меняет длину, потом преобразует обратно в тип string . Например, следующее действие

```
String s = "Это" + " одна " + "строка";
```

компилятор выполнит так:

```
String s = new StringBuffer().append("Это").append(" одна ")
       .append("строка").toString();
```

Будет создан объект класса stringBuffer , в него последовательно добавлены строки "Это", " одна ", "строка", и получившийся объект класса StringBuffer будет приведен к типу String методом toString () .

Напомним, что символы в строках хранятся в кодировке Unicode, в которой каждый символ занимает два байта. Тип каждого символа char .

Класс String

Перед работой со строкой ее следует создать. Это можно сделать разными способами.

Как создать строку

Самый простой способ создать строку — это организовать ссылку типа string на строку-константу:

```
String si = "Это строка.";
```

Если константа длинная, можно записать ее в нескольких строках текстового редактора, связывая их операцией сцепления:

```
String s2 = "Это длинная строка, " +
```

"записанная в двух строках исходного текста";

Замечание

Не забывайте разницу между пустой строкой string s = "" , не содержащей ни одного символа, и пустой ссылкой string s = null, не указывающей ни на какую строку и не являющейся объектом.

Самый правильный способ создать объект с точки зрения ООП — это вызвать его конструктор в операции new. Класс string предоставляет вам девять конструкторов:

- **string()** — создается объект с пустой строкой;
- **string (String str)** — из одного объекта создается другой, поэтому этот конструктор используется редко;
- **string (StringBuf fer str)** — преобразованная коп-ия объекта класса BufferString;
- **string(byte[] byteArray)** — объект создается из массива байтов byteArray;
- **String (char [] charArray)** — объект создается из массива charArray символов Unicode;
- String (byte [] byteArray, int offset, int count) — объект создается из части массива байтов byteArray, начинающейся с индекса offset и содержащей count байтов;
- **String (char [] charArray, int offset, int count)** — то же, но массив состоит из символов Unicode;
- **String(byte[] byteArray, String encoding)** — символы, записанные в массиве байтов, задаются в Unicode-строке, с учетом кодировки encoding ;

- **String(byte[] byteArray, int offset, int count, String encoding)** — то же самое, но только для части массива.

При неправильном заданий индексов offset , count или кодировки encoding возникает исключительная ситуация.

Конструкторы, использующие массив байтов byteArray , предназначены для создания Unicode-строки из массива байтовых ASCII-кодировок символов. Такая ситуация возникает при чтении ASCII-файлов, извлечении информации из базы данных или при передаче информации по сети.

В самом простом случае компилятор для получения двухбайтовых символов Unicode добавит к каждому байту старший нулевой байт. Получится диапазон ' \u0000 ' — ' \u00ff ' кодировки Unicode, соответствующий кодам Latin 1. Тексты на кириллице будут выведены неправильно.

Локализация

Если же на компьютере сделаны местные установки, как говорят на жаргоне "установлена локаль" (locale) (в MS Windows это выполняется утилитой Regional Options в окне Control Panel), то компилятор, прочитав эти установки, создаст символы Unicode, соответствующие местной кодовой странице. В русифицированном варианте MS Windows это обычно кодовая страница CP1251.

Если исходный массив с кириллическим ASCII-текстом был в кодировке CP1251, то строка Java будет создана правильно. Кириллица попадет в свой диапазон '\u0400'—'\u04FF' кодировки Unicode.

Но у кириллицы есть еще, по меньшей мере, четыре кодировки.

- В MS-DOS применяется кодировка CP866.
- В UNIX обычно применяется кодировка KOI8-R.
- На компьютерах Apple Macintosh используется кодировка MacCyrillic.
- Есть еще и международная кодировка кириллицы ISO8859-5;

Например, байт 11100011 (0xE3 в шестнадцатеричной форме) в кодировке CP1251 представляет кириллическую букву Г , в кодировке CP866 — букву У , в кодировке KOI8-R — букву Ц , в ISO8859-5 — букву у , в MacCyrillic — букву г .

Если исходный кириллический ASCII-текст был в одной из этих кодировок, а местная кодировка CP1251, то Unicode-символы строки Java не будут соответствовать кириллице.

В этих случаях используются последние два конструктора, в которых параметром encoding указывается, какую кодовую таблицу использовать конструктору при создании строки.

Листинг .1 показывает различные случаи записи кириллического текста. В нем создаются три массива байто'в, содержащих слово "Россия" в трех кодировках.

- Массив byteCP1251 содержит слово "Россия" в кодировке CP1251.
- Массив byteCP866 содержит слово "Россия" в кодировке CP866.
- Массив byteKOI8R содержит слово "Россия" в кодировке KOI8-R.

Из каждого массива создаются по три строки с использованием трех кодовых таблиц.

Кроме того, из массива символов c[] создается строка s1 , из массива бай-тов, записанного в кодировке CP866, создается строка s2 . Наконец, создается ссылка зз на строку-константу.

Все эти данные выводятся на консоль MS Windows 2000, как показано ниже

В первые три строки консоли выводятся массивы байтов byteCP1251 , byteCP866 и byteKOI8R без преобразования в Unicode. Это выполняется методом write() класса FilterOutputStream из пакета java.io .

В следующие три строки консоли выведены строки Java, полученные из массива символов c[] , массива byteCP866 и строки-константы.

Следующие строки консоли содержат преобразованные массивы.

Вы видите, что на консоль правильно выводится только массив в кодировке CP866, записанный в строку с использованием кодовой таблицы CP1251.

В чем дело? Здесь свой вклад в проблему русификации вносит вывод потока символов на консоль или в файл.

= = = = = = = = = = = = = = = =

Все эти данные выводятся на консоль MS Windows 2000, как показано нижек

В первые три строки консоли выводятся массивы байтов byteCP1251 , byteCP866 и byteKOI8R без преобразования в Unicode. Это выполняется методом write() класса FilterOutputStream из пакета java.io .

В следующие три строки консоли выведены строки Java, полученные из массива символов c[] , массива byteCP866 и строки-константы.

Следующие строки консоли содержат преобразованные массивы.

Вы видите, что на консоль правильно выводится только массив в кодировке CP866, записанный в строку с использованием кодовой таблицы CP1251.

В чем дело? Здесь свой вклад в проблему русификации вносит вывод потока символов на консоль или в файл.

Как видите, кириллица выглядит совсем по-другому. Правильные символы Unicode кириллицы получаются, если использовать ту же кодовую таблицу, в которой записан исходный массив байтов.

Вопросы русификации мы еще будем обсуждать в *главах 9 и 18,* а пока заметьте, что при создании строки из массива байтов лучше указывать ту же самую кириллическую кодировку, в которой записан массив. Тогда вы получите строку Java с правильными символами Unicode.

При выводе же строки на консоль, в окно, в файл или при передаче по сети лучше преобразовать строку Java с символами Unicode по правилам вывода в нужное место.

Еще один способ создать строку — это использовать два статических метода

copyValueOf(chart] charArray) и copyValueOf(char[] charArray, int offset, int length).

Они создают строку по заданному массиву символов и возвращают ее в качестве результата своей работы. Например, после выполнения следующего фрагмента программы

```
chart] c = ('С', 'и', 'м', 'в', 'о' , 'л', 'ь', 'н', 'ы', 'й'};
String s1 = String.copyValueOf(c);
String s2 = String.copyValueOf(c, 3, 7);
```

получим в объекте s1 строку " Символьный ", а в объекте s2 — строку " вольный ".

Сцепление строк

Со строками можно производить операцию *сцепления строк* (concatenation), обозначаемую знаком плюс +. Эта операция создает новую строку, просто составленную из состыкованных первой и второй строк, как показано в начале данной главы. Ее можно применять и к константам, и к переменным. Например:

```
String attention = "Внимание: ";
String s = attention + "неизвестный символ";
```

Вторая операция — присваивание += — применяется к переменным в левой части:

```
attention += s;
```

Поскольку операция + перегружена со сложения чисел на сцепление строк, встает вопрос о приоритете этих операций. У сцепления строк приоритет выше, чем у сложения, поэтому, записав "2" + 2 + 2 , получим строку " 222 ". Но, записав 2 + 2 + "2" , получим строку "42", поскольку действия выполняются слева направо. Если же запишем "2" + (2 + 2) , то получим "24" .

Манипуляции строками

В классе string есть множество методов для работы со строками. Посмотрим, что они позволяют делать.

Как узнать длину строки

Для того чтобы узнать длину строки, т. е. количество символов в ней, надо обратиться к методу length() :

```
String s = "Write once, run anywhere.";
int len = s.length{);
```

или еще проще

```
int len = "Write once, run anywhere.".length();
```

поскольку строка-константа — полноценный объект класса string . Заметьте, что строка — это не массив, у нее нет поля length .

Внимательный читатель, изучивший рис. 4.7, готов со мной не согласиться. Ну, что же, действительно, символы хранятся в массиве, но он закрыт, как и все поля класса string .

Как выбрать символы из строки

Выбрать символ с индексом ind (индекс первого символа равен нулю) можно методом charAt(int ind) Если индекс ind отрицателен или не меньше чем длина строки, возникает исключительная ситуация. Например, после определения

```
char ch = s.charAt(3);
```

переменная ch будет иметь значение 't'

Все символы строки в виде массива символов можно получить методом

toCharArray() , возвращающим массив символов.

Если же надо включить в массив символов dst , начиная с индекса ind массива подстроку от индекса begin включительно до индекса end исключительно, то используйте метод getChars(int begin, int end, char[] dst, int ind) типа void .

В массив будет записано end - begin символов, которые займут элементы массива, начиная с индекса ind до индекса in d + (end - begin) - 1 .

Этот метод создает исключительную ситуацию в следующих случаях:

- ссылка dst = null ;
- индекс begin отрицателен;
- индекс begin больше индекса end ;
- индекс end больше длины строки;
- индекс ind отрицателен;
- ind + (end — begin) > dst.length.

Например, после выполнения

```
char[] ch = ('К', 'о', 'р', 'о', 'л', 'ь', ' ', 'л', 'е', 'т', 'а'};
"Пароль легко найти".getChars(2, 8, ch, 2);
```

результат будет таков:

```
ch = ('К', 'о', 'р', 'о', 'л', 'ь ', ' ', 'л', 'е', 'т', 'а'};
```

Если надо получить массив байтов, содержащий все символы строки в байтовой кодировке ASCII, то используйте метод getBytes() .

Этот метод при переводе символов из Unicode в ASCII использует локальную кодовую таблицу.

Если же надо получить массив байтов не в локальной кодировке, а в какой-то другой, используйте метод getBytes(String encoding) .

Так сделано в листинге 5.1 при создании объекта msg. Строка "\'Госсия в\"" перекодировалась в массив CP866-байтов для правильного вывода кириллицы в консольное окно **Command Prompt** операционной системы Windows 2000.

Как выбрать подстроку

Метод substring(int begin, int end) выделяет подстроку от символа с индексом begin включительно до символа с индексом end исключительно. Длина подстроки будет равна end - begin .

Метод substring (int begin) выделяет подстроку от индекса begin включительно до конца строки.

Если индексы отрицательны, индекс end больше длины строки или begin больше чем end , то возникает исключительная ситуация.

Например, после выполнения

```
String s = "Write once, run anywhere.";
String sub1 = s.substring(6, 10);
String sub2 = s.substring(16);
```

получим в строке sub1 значение " once ", а в sub2 — значение " anywhere ".

Как сравнить строки

Операция сравнения == сопоставляет только ссылки на строки. Она выясняет, указывают ли ссылки на одну и ту же строку. Например, для строк

```
String s1 = "Какая-то строка";
String s2 = "Другая-строка";
```

сравнение s1 == s2 дает в результате false .

Значение true получится, только если обе ссылки указывают на одну и ту же строку, например, после присваивания si = s2 .

Интересно, что если мы определим s2 так:

```
String s2 == "Какая-то строка";
```

то сравнение s1 == s2 даст в результате true , потому что компилятор создаст только один экземпляр константы "Какая-то строка" и направит на него все ссылки.

Вы, разумеется, хотите сравнивать не ссылки, а содержимое строк. Для этого есть несколько методов.

Логический метод equals (object obj) , переопределенный из класса object , возвращает true , если аргумент obj не равен null , является объектом класса string , и строка, содержащаяся в нем, полностью

идентична данной строке вплоть до совпадения регистра букв. В остальных случаях возвращается значение false .

Логический метод equalsIgnoreCase(object obj) работает так же, но одинаковые буквы, записанные в разных регистрах, считаются совпадающими.

Например, s2.equals("другая строка") даст в результате false , а s2.equalsIgnoreCase("другая строка") возвратит true .

Метод compareTo(string str) возвращает целое число типа int , вычисленное по следующим правилам:

1. Сравниваются символы данной строки this и строки str с одинаковым индексом, пока не встретятся различные символы с индексом, допустим k , или пока одна из строк не закончится.
2. В первом случае возвращается значение this.charAt(k) - str.charAt(k), т. е. разность кодировок Unicode первйх несовпадающих символов.
3. Во втором случае возвращается значение this.length() - str.length() , т. е. разность длин строк.
4. Если строки совпадают, возвращается 0.

Если значение str равно null , возникает исключительная ситуация.

Нуль возвращается в той же ситуации, в которой метод equals() возвращает true .

Метод compareToignoreCase(string str) производит сравнение без учета регистра букв, точнее говоря, выполняется метод

```
this.toUpperCase().toLowerCase().compareTo(
str.toUpperCase().toLowerCase());
```

Еще один метод— compareTo (Object obj) создает исключительную ситуацию, если obj не является строкой. В остальном он работает как метод compareTo(String str).

Эти методы не учитывают алфавитное расположение символов в локальной кодировке.

Русские буквы расположены в Unicode по алфавиту, за исключением одной буквы. Заглавная буква Ё расположена перед всеми кириллическими буквами, ее код '\ u040l ', а строчная буква е — после всех русских букв, ее код '\ u0451 '.

Если вас такое расположение не устраивает, задайте свое размещение букв с помощью класса RuleBasedCollator из пакета java.text .

Сравнить подстроку данной строки this с подстрокой той же длины len другой строки str можно логическим методом

```
regionMatches(int indl, String str, int ind2, int len)
```

Здесь ind1 — индекс начала подстроки данной строки this, ind2 — индекс начала подстроки другой строки str . Результат false получается в следующих случаях:

- хотя бы один из индексов ind1 или ind2 отрицателен;
- хотя бы одно из ind1 + len или ind2 + len больше длины соответствующей строки;
- хотя бы одна пара символов не совпадает.

Этот метод различает символы, записанные в разных регистрах. Если надо сравнивать подстроки без учета регистров букв, то используйте логический метод:

```
regionMatches(boolean flag, int indl, String str, int ind2, int len)
```

Если первый параметр flag равен true , то регистр букв при сравнении подстрок не учитывается, если false — учитывается.

Как найти символ в строке

Поиск всегда ведется с учетом регистра букв.

Первое появление символа ch в данной строке this можно отследить методом indexOf(int ch) , возвращающим индекс этого символа в строке или -1 , если символа ch в строке this нет.

Например, "Молоко", indexOf('o') выдаст в результате 1 .

Конечно, этот метод выполняет в цикле последовательные сравнения this.charAt(k++> == ch , пока не получит значение true .

Второе и следующие появления символа ch в данной строке this можно отследить методом indexOf(int ch, int ind) .

Этот метод начинает поиск символа ch с индекса ind . Если ind < o, то поиск идет с начала строки, если ind больше длины строки, то символ не ищется, т. е. возвращается -1.

Например, "молоко".indexof('o', indexof ('o') + 1) даст в результате 3. .

Последнее появление символа ch в данной строке this отслеживает метод lastIndexof (int ch). Он просматривает строку в обратном порядке. Если символ ch не найден, возвращается.-1.

Например, "Молоко".lastindexof('o') даст в результате 5.

Предпоследнее и предыдущие появления символа ch в данной строке this можно отследить методом lastIndexof (int ch, int ind) , который просматривает строку в обратном порядке, начиная с индекса ind .

Если ind больше длины строки, то поиск идёт от конца строки, если ind < o, то возвращается-1.

Как найти подстроку

Поиск всегда ведется с учетом регистра букв.

Первое вхождение подстроки sub в данную строку this отыскивает метод indexof (String sub). Он возвращает индекс первого символа первого вхождения подстроки sub в строку или -1, если подстрока sub не входит в строку this . Например, " Раскраска ".indexof ("рас") даст в результате 4.

Если вы хотите начать поиск не с начала строки, ас какого-то индекса ind , используйте метод indexOf (String sub, int ind). если i nd < 0 , то поиск идет с начала строки, если ind больше .длины строки, то символ не ищется, т. е. возвращается -1.

Последнее вхождение подстроки sub в данную строку this можно отыскать методом lastindexof (string sub), возвращающим индекс первого символа последнего вхождения подстроки sub в строку this или (-1), если подстрока sub не входит в строку this .

Последнее вхождение подстроки sub не во всю строку this , а только в ее начало до индекса ind можно отыскать методом l astIndexof(String stf, int ind). Если ind больше длины строки, то .поиск идет от конца строки, если ind < o , то возвращается -1.

Для того чтобы проверить, не начинается ли данная строка this с подстроки sub , используйте логический метод startsWith(string sub) , возвращающий true , если данная строка this начинается с подстроки sub , или совпадает с ней, или подстрока sub пуста.

Можно проверить и появление подстроки sub в данной строке this , начиная с некоторого индекса ind логическим методом s tartsWith(String sub),int ind). Если индекс ind отрицателен или больше длины строки, возвращается false .

Для того чтобы проверить, не заканчивается ли данная строка this подстрокой sub , используйте логический метод endsWitht(String sub) . Учтите, что он возвращает true , если подстрока sub совпадает со всей строкой или подстрока sub пуста.

Например, if (fileName.endsWith(". Java")) отследит имена файлов с исходными текстами Java.

Перечисленные выше методы создают исключительную ситуацию, если

sub == null.

Если вы хотите осуществить поиск, не учитывающий регистр букв, измените предварительно регистр всех символов строки.

Как изменить регистр букв

Метод toLowerCase () возвращает новую строку, в которой все буквы переведены в нижний регистр, т. е. сделаны строчными.

Метод toUpperCase () возвращает новую строку, в которой все буквы переведены в верхний регистр, т. е. сделаны прописными.

При этом используется локальная кодовая таблица по умолчанию. Если нужна другая локаль, то применяются методы toLowerCase(Locale l oc) и toUpperCase(Locale loc).

Как заменить отдельный символ

Метод replace (int old, int new) возвращает новую строку, в которой все вхождения символа old заменены символом new . Если символа old в строке нет, то возвращается ссылка на исходную строку.

Например, после выполнения " Рука в руку сует хлеб" , replace ('у', 'е') получим строку " Река в реке сеет хлеб".

Регистр букв при замене учитывается

Как убрать пробелы в начале и конце строки

Метод trim o возвращает новую строку, в которой удалены начальные и конечные символы с кодами, не превышающими '\u0020 '.

Как преобразовать данные другого типа в строку

В языке Java принято соглашение — каждый класс отвечает за преобразование других типов в тип этого класса и должен содержать нужные для этого методы.

Класс string содержит восемь статических методов valueof (type elem) преобразования В строку примитивных типов boolean, char, int, long, float, double , массива char[] , и просто объекта типа object .

Девятый метод valueof(char[] ch, int offset, int len) преобразует в строку подмассив массива ch , начинающийся с индекса offset и имеющий len элементов.

Кроме того, в каждом классе есть метод tostring () , переопределенный или просто унаследованный от класса Object . Он преобразует объекты класса в строку. Фактически, метод valueOf о вызывает метод tostring() соответствующего класса. Поэтому результат преобразования зависит от того, как реализован метод tostring ().

Еще один простой способ — сцепить значение elem какого-либо типа с пустой строкой: "" + elem. При этом неявно вызывается метод elem. toString ().

Класс StringBuffer

Объекты класса StringBuffer — это строки переменной длины. Только что созданный объект имеет буфер определенной *емкости* (capacity), по умолчанию достаточной для хранения 16 символов. Емкость можно задать в конструкторе объекта.

Как только буфер начинает переполняться, его емкость автоматически увеличивается, чтобы вместить новые символы.

В любое время емкость буфера можно увеличить, обратившись к методу ensureCapacity(int minCapacity)

Этот метод изменит емкость, только если minCapacity будет больше длины хранящейся в объекте строки. Емкость будет увеличена по следующему правилу. Пусть емкость буфера равна N. Тогда новая емкость будет равна

```
Max(2 * N + 2, minCapacity)
```

Таким образом, емкость буфера нельзя увеличить менее чем вдвое.

Методом setLength (int newLength) можно установить любую длину строки.

Если она окажется больше текущей длины, то дополнительные символы будут равны ' \uOOOO' . Если она будет меньше текущей длины, то строка будет обрезана, последние символы потеряются, точнее, будут заменены символом '\uOOOO' . Емкость при этом не изменится.

Если число newLength окажется отрицательным, возникнет исключительная ситуация.

Совет

Будьте осторожны, устанавливая новую длину объекта.

Количество символов в строке можно узнать, как и для объекта класса String , методом length() , а емкость — методом capacity ().

Создать объект класса stringBuf fer можно только конструкторами.

Конструкторы

В классе stringBuffer три конструктора:

stringBuffer () — создает пустой объект с емкостью 16 символов;

stringBuffer .(int capacity) — создает пустой объект заданной емкости capacity ;

StringBuffer (String str) — создает объект емкостью str . length () + 16, содержащий строку str .

Как добавить подстроку

В классе stringBuffer есть десять методов append (), добавляющих подстроку в конец строки. Они не создают новый экземпляр строки, а возвращают ссылку на ту же самую, но измененную строку.

Основной метод append (string str) присоединяет строку str в конец данной строки. Если ссылка str == null, то добавляется строка "null".

Шесть методов append (type elem) добавляют примитивные типы boolean, char, int, long, float, double, преобразованные в строку.

Два метода присоединяют к строке массив str и подмассив sub символов,

преобразованные в строку: append (char [] str) И append (char [.] , sub, int offset, int len).

Десятый метод добавляет просто объект append (Object obj). Перед этим объект obj преобразуется в строку своим методом tostring ().

Как вставить подстроку

Десять методов insert () предназначены для вставки строки, указанной параметром метода, в данную строку. Место вставки задается первым параметром метода ind . Это индекс элемента строки, перед которым будет сделана вставка. Он должен быть неотрицательным и меньше длины строки, иначе возникнет исключительная ситуация. Строка раздвигается, емкость буфера при необходимости увеличивается. Методы возвращают ссылку ни ту же преобразованную строку.

Основной метод insert (int ind, string str) вставляет строку str в данную строку перед ее символом с индексом and . Если ссылка s tr == null вставляется строка "null".

Например, после выполнения

```
String s = new StringBuffer("Это большая строка"). insert(4, "не").toString();
```

ПОЛУЧИМ s == "Это небольшая строка".

Метод sb.insert(sb.length o, "xxx") будет работать так же, как метод sb.append("xxx") .

Шесть методов insert (int ind, type elem) вставляют примитивные типы boolean, char, int, long, float, double, преобразованные в строку.

Два метода вставляют массив str и подмассив sub символов, преобразованные в строку:

```
insert(int ind, chart] str)
insert(int ind, char[] sub, int offset, int len)
```

Десятый метод вставляет просто объект :

```
insert(int ind, Object obj)
```

Объект obj перед добавлением преобразуется в строку своим методом toString ().

Как удалить подстроку

Метод delete tint begin, int end) удаляет из строки символы, начиная с индекса begin включительно до индекса end исключительно, если end больше длины строки, то до конца строки.

Например, после выполнения

```
String s = new StringBuffer("Это небольшая строка").
    delete(4, 6).toString();
```

получим s == "Это большая строка".

Если begin отрицательно, больше длины строки или больше end , возникает исключительная ситуация.

Если begin == end, удаление не происходит.

Как удалить символ

Метод deieteCharAt (int ind) удаляет символ с указанным индексом ind . Длина строки уменьшается на единицу.

Если индекс ind отрицателен или больше длины строки, возникает исключительная ситуация.

Как заменить подстроку

Метод replace (int begin, int end. String str) удаляет символы из строки, начиная с индекса begin включительно до индекса end исключительно, если end больше длины строки, то до конца строки, и вставляет вместо них строку str .

Если begin отрицательно, больше длины строки или больше end , возникает исключительная ситуация.

Разумеется, метод replace () — это последовательное выполнение методов

delete () и insert ().

Как перевернуть строку

Метод reverse() меняет порядок расположения символов в строке на обратный порядок.

Например, после выполнения

```
String s = new StringBuffer("Это небольшая строка"),
      reverse().toString();
```

получим s == "акортс яашьлобен отЭ".

Синтаксический разбор строки

Задача разбора введенного текста — *парсинг* (parsing) — вечная задача программирования, наряду с сортировкой и поиском. Написана масса программ-парсеров (parser), разбирающих текст по различным признакам. Есть даже программы, генерирующие парсеры по заданным правилам разбора: YACC, LEX и др.

Но задача остается. И вот очередной программист, отчаявшись найти что-нибудь подходящее, берется за разработку собственной программы разбора.

В пакет java.utii входит простой класс stringiokenizer , облегчающий разбор строк.

Класс StringTokenizer

Класс StringTokenizer из пакета java.utii небольшой, в нем три конструктора и шесть методов.

Первый конструктор StringTokenizer (String str) создает объект, готовый разбить строку str на слова, разделенные пробелами, символами табуляций '\t', перевода строки '\n' и возврата каретки '\r' . Разделители не включаются в число слов.

Второй конструктор StringTokenizer (String str. String delimeters) задает разделители вторым параметром deiimeters , например:

```
StringTokenizer("Казнить,нельзя:пробелов-нет", " \t\n\r,:-");
```

Здесь первый разделитель — пробел. Потом идут символ табуляции, символ перевода строки, символ возврата каретки, запятая, двоеточие, дефис. Порядок расположения разделителей в строке deiimeters не имеет значения. Разделители не включаются в число слов.

Третий конструктор позволяет включить разделители в число слов:

```
StringTokenizer(String str, String deiimeters, boolean flag);
```

Если параметр flag равен true , то разделители включаются в число слов, если false — нет. Например:

```
StringTokenizer("a - (Ь + с) / Ь * с", " \t\n\r+*-/(), true);
```

В разборе строки на слова активно участвуют два метода:

метод nextToken () возвращает в виде строки следующее слово;

логический метод hasMoreTokens () возвращает true , если в строке еще есть слова, и false , если слов больше нет.

Третий метод countTokens () возвращает число оставшихся слов.

Четвертый метод nextToken(string newDeiimeters) позволяет "на ходу" менять разделители. Следующее слово будет выделено по новым разделителям newDeiimeters; новые разделители действуют далее вместо старых разделителей, определенных в конструкторе или предыдущем методе nextToken () .

Оставшиеся два метода nextEiement () и hasMoreEiements () реализуют интерфейс Enumeration . Они просто обращаются к методам nextToken () и hasMoreTokens().

Схема очень проста (листинг .2).

Листинг.2. Разбиение строки на слова :

```
String s = "Строка, которую мы хотим разобрать на слова";
StringTokenizer st = new StringTokenizer(s, " \t\n\r,.");
while(st.hasMoreTokens()){
// Получаем слово и что-нибудь делаем с ним, например,
// просто выводим на экран
System.out.println(st.nextToken()) ;
}
```

Полученные слова обычно заносятся в какой-нибудь класс-коллекцию: Vector, Stack или другой, наиболее подходящий для дальнейшей обработки текста контейнер. Классы-коллекции мы рассмотрим в следующей главе.

Заключение

Все методы представленных в этой главе классов написаны на языке Java. Их исходные тексты можно посмотреть, они входят в состав JDK.

Листинг Создание кириллических строк

```
/* Текст подготовлен к кодировке ВИНДУЗЫ */
class StringTest{
  public static void main(String[] args){
    System.out.println("В винузе" );
    String winLikeWin  = null, winLikeDOS  = null, winLikeUNIX = null;
    String dosLikeWin  = null, dosLikeDOS  = null, dosLikeUNIX = null;
    String unixLikeWin = null, unixLikeDOS = null, unixLikeUNIX = null;
    String msg = null;
    byte[] byteCp1251 = {
     (byte)0xD0, (byte)0xEE, (byte)0xFl,
     (byte)0xFl, (byte)0xE8, (byte)0xFF
    };
    byte[] byteCp866 = {
     (byte)0x90, (byte)0xAE, (byte)0xE1,
     (byte)0xE1, (byte)0xA8, (byte)0xEF
    };
    byte[] byteKOISR = {
```

```
 (byte)0xF2, (byte)0xCF, (byte)0xD3,
 (byte)0xD3, (byte)0xC9, (byte)0xDl
};
char[] c = {'Р', 'о', 'с', 'с', 'и', 'я'};
String s1 = new String(c);
String s2 = new String(byteCp866);   // "<п ЄR-бR<Ё MS Windows
String s3 = "_RббЁп";
System.out.println();
try{
      // 'RRЎй_-Ё_ ў Cp866 ¤<п ўлўR¤  -  ЄR-бR<м MS Windows.
      // Сообщение в Cp866 для вывода на консоль MS Windows.
      msg = new String("\"Россия\" в ".getBytes("Cp866"), "Cp1251");

      winLikeWin   = new String(byteCp1251, "Cp1251");  //_a ўЁ<м-R
      winLikeDOS   = new String(byteCp1251, "Cp866" );
      winLikeUNIX  = new String(byteCp1251, "KOI8-R");
      dosLikeWin   = new String(byteCp866,  "Cp1251");  // "<п ЄR-бR<Ё
      dosLikeDOS   = new String(byteCp866,  "Cp866" );   // _a ўЁ<м-R
      dosLikeUNIX  = new String(byteCp866,  "KOI8-R");
      unixLikeWin  = new String(byteKOISR,  "Cp1251");
      unixLikeDOS  = new String(byteKOISR,  "Cp866" );
      unixLikeUNIX = new String(byteKOISR,  "KOI8-R");  // _a ўЁ<м-R
      System.out.print(msg + "Cp1251: ");
      System.out.write(byteCp1251);
      System.out.println();
      System.out.print(msg + "Cp866 : ");
      System.out.write(byteCp866);
      System.out.println();
      System.out.print(msg + "KOI8-R: ");
      System.out.write(byteKOISR);
}
catch(Exception e) {
  e.printStackTrace();
}
System.out.println();
System.out.println();
System.out.println(msg + "char array       : " + s1);
System.out.println(msg + "default encoding : " + s2);
System.out.println(msg + "string constant  : " + s3);
System.out.println();
System.out.println(msg + "Cp1251 -> Cp1251: " + winLikeWin);
System.out.println(msg + "Cp1251 -> Cp866 : " + winLikeDOS);
System.out.println(msg + "Cp1251 -> KOI8-R: " + winLikeUNIX);
System.out.println(msg + "Cp866  -> Cp1251: " + dosLikeWin);
System.out.println(msg + "Cp866  -> Cp866 : " + dosLikeDOS);
System.out.println(msg + "Cp866  -> KOI8-R: " + dosLikeUNIX);
System.out.println(msg + "KOI8-R -> Cpl251: " + unixLikeWin);
System.out.println(msg + "KOI8-R -> Cp866 : " + unixLikeDOS);
System.out.println(msg + "KOI8-R -> KOI8-R: " + unixLikeUNIX);
  }
}
```

Результат на консоль

```
C:\Test\java\win>java StringTest
┬ тшэєчх
```

```
"Россия" в Cp1251: ╨ю☼☼ш
"Россия" в Cp866 : Россия
"Россия" в KOI8-R: Є╧╨╨╔

"Россия" в char array      : ╨юёёш
"Россия" в default encoding : Россия
"Россия" в string constant  : Россия

"Россия" в Cp1251 -> Cp1251: ╨ю☼☼ш
"Россия" в Cp1251 -> Cp866 : ?■☼☼°a
"Россия" в Cp1251 -> KOI8-R: я═☼☼ ╔ ┌
"Россия" в Cp866  -> Cp1251: Россия
"Россия" в Cp866  -> Cp866 : ╨юёёш
"Россия" в Cp866  -> KOI8-R: ?? └ └?╬
"Россия" в KOI8-R -> Cpl251: Є╧╨╨╔
"Россия" в KOI8-R -> Cp866 : к????
"Россия" в KOI8-R -> KOI8-R: ╨юёёш

C:\Test\java\win>
```

/* текст подготовлен в кодировке CMD */

```
class StringTest{
  public static void main(String[] args){
    System.out.println("АПРОЛ-апрол" );
    String winLikeWin  = null, winLikeDOS  = null, winLikeUNIX = null;
    String dosLikeWin  = null, dosLikeDOS  = null, dosLikeUNIX = null;
    String unixLikeWin = null, unixLikeDOS = null, unixLikeUNIX = null;
    String msg = null;
    byte[] byteCp1251 = {
     (byte)0xD0, (byte)0xEE, (byte)0xF1,
     (byte)0xF1, (byte)0xE8, (byte)0xFF
    };
    byte[] byteCp866 = {
     (byte)0x90, (byte)0xAE, (byte)0xE1,
     (byte)0xEl, (byte)0xA8, (byte)0xEF
    };
    byte[] byteKOISR = {
     (byte)0xF2, (byte)0xCF, (byte)0xD3,
     (byte)0xD3, (byte)0xC9, (byte)0xDl
    };
    char[] c = {'Р', 'о', 'с', 'с', 'и', 'я'};
    String s1 = new String(c);
    String s2 = new String(byteCp866);   // Для консоли MS Windows
    String s3 = "Россия";
    System.out.println();
    try{
          // Сообщение в Cp866 для вывода на консоль MS Windows.
          //msg = new String("\"Россия\" в ".getBytes("Cp866"), "Cp866"); //"Cp1251");
          msg = new String("\"Россия\" в "); //"Cp1251");

          winLikeWin   = new String(byteCp1251, "Cp1251");  //Правильно
          winLikeDOS   = new String(byteCp1251, "Cp866" );
          winLikeUNIX  = new String(byteCp1251, "KOI8-R");
          dosLikeWin   = new String(byteCp866,  "Cp1251");  // Для консоли
          dosLikeDOS   = new String(byteCp866,  "Cp866" );   // Правильно
```

```
        dosLikeUNIX  = new String(byteCp866,  "KOI8-R");
        unixLikeWin  = new String(byteKOISR,  "Cp1251");
        unixLikeDOS  = new String(byteKOISR,  "Cp866" );
        unixLikeUNIX = new String(byteKOISR,  "KOI8-R");  // Правильно
        System.out.print(msg + "Cp1251: ");
        System.out.write(byteCp1251);
        System.out.println();
        System.out.print(msg + "Cp866 : ");
        System.out.write(byteCp866);
        System.out.println();
        System.out.print(msg + "KOI8-R: ");
        System.out.write(byteKOISR);
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    System.out.println();
    System.out.println();
    System.out.println(msg + "char array       : " + s1);
    System.out.println(msg + "default encoding : " + s2);
    System.out.println(msg + "string constant  : " + s3);
    System.out.println();
    System.out.println(msg + "Cp1251 -> Cp1251: " + winLikeWin);
    System.out.println(msg + "Cp1251 -> Cp866 : " + winLikeDOS);
    System.out.println(msg + "Cp1251 -> KOI8-R: " + winLikeUNIX);
    System.out.println(msg + "Cp866  -> Cp1251: " + dosLikeWin);
    System.out.println(msg + "Cp866  -> Cp866 : " + dosLikeDOS);
    System.out.println(msg + "Cp866  -> KOI8-R: " + dosLikeUNIX);
    System.out.println(msg + "KOI8-R -> Cpl251: " + unixLikeWin);
    System.out.println(msg + "KOI8-R -> Cp866 : " + unixLikeDOS);
    System.out.println(msg + "KOI8-R -> KOI8-R: " + unixLikeUNIX);
  }
}
```

Результат на консоль

```
C:\Test\java>java StringTest
АПРОЛ-апрол

"Россия" в Cp1251: ╨юёёш
"Россия" в Cp866 : Россия
"Россия" в KOI8-R: Є╧╥╥╔

"Россия" в char array       : Россия
"Россия" в default encoding : Россия
"Россия" в string constant  : Россия

"Россия" в Cp1251 -> Cp1251: ╨юёёш
"Россия" в Cp1251 -> Cp866 : ?■╕╕ °a
"Россия" в Cp1251 -> KOI8-R: я═▀▀ ╔ ╔
"Россия" в Cp866  -> Cp1251: Россия
"Россия" в Cp866  -> Cp866 : ╨юёёш
"Россия" в Cp866  -> KOI8-R: ?? └ └?╨
"Россия" в KOI8-R -> Cpl251: Є╧╥╥╔
"Россия" в KOI8-R -> Cp866 : к????
"Россия" в KOI8-R -> KOI8-R: ╨юёёш
C:\Test\java>
```