Работа со строками

Строки тоже являются переменными ссылочного типа, а точнее — ссылками на объекты одного из нескольких строковых классов Java. Мы рассмотрим класс String.

Самый распространенный способ создать строку — это организовать ссылку типа String на строку-константу:

```
String s = "Это строка";
```

Можно просто сначала объявить переменную (которая получит значение null), а потом заставить ее ссылаться на строку-константу, другую строку или воспользоваться командой new, чтобы явным образом выделить память для строки:

```
String s1, s2, s3; // Объявление трех переменных, которые пока не связаны ни с какой строкой
s1 = "Да здравствует день танкиста"; // Переменная s1 теперь ссылается на область памяти, в которой хранится строка "Да здравствует день танкиста"
s2 = s1; // Теперь обе переменные s1 и s2 ссылаются на одно и то же место памяти
s3 = new String(); // s3 ссылается на место в памяти, где хранится пустая строка
```

Работа со строками

Объединение (сцепление) строк

Сцепление строк производится командой **+**. Ей соответствует оператор +=, который часто бывает очень удобен:

```
String S = "Превед";
String S1 = "медвед";
S += ", " + S1 + "!"; // Теперь S ссылается на строку “Превед, медвед!”
```

Длина строки

Определить длину строки можно методом **length**():

```
int x = S.length(); // Переменная x получит значение 15
```

Обратите внимание, String является классом, а length() - его методом, и поэтому указывается через точку после имени переменной. Аналогично записываются и другие методы класса String.

Работа со строками

Получение отдельных символов строки

Метод **charAt(int i)** возвращает символ строки с индексом i. Индекс первого символа строки — 0 (т.е. символы строки индексируются (нумеруются) аналогично элементам массива. Например:

```
char ch = S.charAt(2); // Переменная ch будет иметь значение 'e'
```

Метод **toCharArray**() преобразует строку в массив символов:

```
char[] ch1 = S.toCharArray(); // ch1 будет иметь представлять собой массив {'П','р','е','в','е','д'
,',',' ','м','е','д','в','е','д','!'}
```

Замена отдельного символа

Метод **replace(char old, char new)** возвращает новую строку, в которой все вхождения символа old заменены на символ new.

```
String SS = S.replace('е', 'ю'); // SS = "Прювюд, мюдвюд!"
```

Работа со строками

Получение подстроки

Метод **substring(int begin, int end)** возвращает фрагмент исходной строки от символа с индексом begin до символа с индексом end-1 включительно. Если не указывать end, будет возвращен фрагмент исходной строки, начиная с символа с индексом begin и до конца:

```
String S2 = S.substring(4,7); // S2 = "ед,"
S2 = S.substring(4); // S2 = "ед, медвед!"
```

Разбиение строки на подстроки

Метод **split(String regExp)** разбивает строку на фрагменты, используя в качестве разделителей символы, входящие в параметр regExp, и возвращает ссылку на массив, составленный из этих фрагментов. Сами разделители ни в одну подстроку не входят.

```
String parts[] = S.split(" "); // Разбили строку S на отдельные слова, используя пробел в качестве разделителя, в результате получили массив parts, где parts[0]=Превед,; parts[1]=медвед!;
String parts1[] = S.split("e| |\\,"); // Разбили строку S на отдельные слова, используя в качестве разделителя пробел или букву е или «,», в результате parts[0]=Пр; parts[1]=в; parts[2]=д; parts[3]=; parts[4]=м; parts[5]=дв; parts[6]=д!;
```

Работа со строками

Сравнение строк

Если сравнивать строки, используя логическую операцию ==, то ее результатом будет true только в том случае, если строковые переменные указывают (ссылаются) на один и тот же объект в памяти.

Если же необходимо проверить две строки на совпадение, следует использовать стандартный метод **equals(Object obj)**. Он возвращает true, если две строки являются полностью идентичными вплоть до регистра букв, и false в противном случае. Его следует использовать следующим образом:

```
S1.equals(S2); // Вернет true, если строки S1 и S2 идентичны
S2.equals(S1); // Абсолютно то же самое

boolean b = S.equals("Превед, медвед!"); // b = true
```

Метод **equalsIgnoreCase** (Object obj) работает аналогично, но строки, записанные в разных регистрах, считает совпадающими.

Работа со строками

Поиск подстроки

Метод **indexOf (char ch)** возвращает индекс первого вхождения символа ch в исходную строку. Если задействовать этот метод в форме **indexOf *(*char ch, init i)**, то есть указать два параметра при вызове, то поиск вхождения начнется с символа с индексом i. Если такого символа в строке нет, результатом будет -1.

```
int pos = S.indexOf('в'); // pos = 3
pos = "Вася".indexOf('с'); // pos = 2
pos = "Корова".indexOf('о', 2); // pos = 3
pos = "Корова".indexOf('К', 2); // pos = -1, поиск ведется с
учетом регистра
```

Последнее вхождение символа можно найти с помощью метода **lastIndexOf(char ch)** или **lastIndexOf(char ch, int i)**, который работает аналогично, но просматривает строку с конца.

Работа со строками

У всех перечисленных методов есть одноименные варианты, которые принимают в качестве параметра строку вместо символа и проверяют, содержится ли эта строка в исходной строке.

```
pos = "Корова".indexOf("ор"); // pos = 1
pos = "Барабанщик барабанил в
барабан".indexOf("барабан", 5); // pos = 11
pos = "Корова".indexOf("Вася"); // pos = -1
```

Изменение регистра символов в строке

Метод **toLowerCase**() возвращает новую строку, в которой все буквы сделаны строчными. Метод **toUpperCase**() возвращает новую строку, в которой все буквы сделаны прописными.

```
S = S.toUpperCase(); // S = "ПРЕВЕД, МЕДВЕД!"
```

Работа со строками

Статические строки

Статические строки реализуются через класс String. Объект строки автоматически создается при использовании строковой литералы. А также для строк доступна операция +, позволяющая соединить несколько строк в одну. Если один из операндов не строка, то он автоматически преобразуется в строку. Для объектов в этих целях используется метод toString().

Работа со строками

compareTo(String anotherString)	лексиграфическое сравнение строк;
compareToIgnoreCase(String str)	лексиграфическое сравнение строк без учета регистра символов;
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	тест на идентичность участков строк, можно указать учет регистра символов;
regionMatches(int toffset, String other, int ooffset, int len)	тест на идентичность участков строк;
concat(String str)	возвращает соединение двух строк;
contains(CharSequence s)	проверяет, входит ли указанная последовательность символов в строку;

Работа со строками

endsWith(String suffix)	проверяет завершается ли строка указанным суффиксом;
startsWith(String prefix)	проверяет, начинается ли строка с указанного префикса;
startsWith(String prefix, int toffset)	проверяет, начинается ли строка в указанной позиции с указанного префикса;
equals(Object anObject)	проверяет идентична ли строка указанному объекту;
getBytes()	возвращает байтовое представление строки;
getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	возвращает символьное представление участка строки;
hashCode()	хеш код строки;

Работа со строками

indexOf(int ch)	поиск первого вхождения символа в строке;
indexOf(int ch, int fromIndex)	поиск первого вхождения символа в строке с указанной позиции;
indexOf(String str)	поиск первого вхождения указанной подстроки;
indexOf(String str, int fromIndex)	поиск первого вхождения указанной подстроки с указанной позиции;
lastIndexOf(int ch)	поиск последнего входения символа;
lastIndexOf(int ch, int fromIndex)	поиск последнего входения символа с указанной позиции;
lastIndexOf(String str)	поиск последнего вхождения строки;

Работа со строками

lastIndexOf(String str, int fromIndex)	поиск последнего вхождения строки с указанной позиции;
replace(char oldChar, char newChar)	замена в строке одного символа на другой;
replace(CharSequence target, CharSequence replacement)	замена одной подстроки другой;
substring(int beginIndex, int endIndex)	возвратить подстроку как строку;
toLowerCase()	преобразовать строку в нижний регистр;
toLowerCase(Locale locale)	преобразовать строку в нижний регистр, используя указанную локализацию;

Работа со строками

toUpperCase()	преобразовать строку в верхний регистр;
toUpperCase(Locale locale)	преобразовать строку в верхний регистр, используя указанную локализацию;
trim()	отсечь на концах строки пустые символы;
valueOf(a)	статические методы преобразования различных типов в строку.

Методы поиска возвращают индекс вхождения или -1 если искомое не найдено. Методы преобразования как replace не изменяют саму строку а возвращают соответствующий новый объект строки.

Работа со строками

Динамические строки

Если необходимо сделать множество преобразований над строкой, то на это время эффективнее воспользоваться динамической строкой, реализуемой классом StringBuffer.

append(A)	преобразовать A в строку и добавить в конец;
insert(int offset, A)	преобразовать A в строку и вставить ее в указанную позицию;
delete(int start, int end)	удалить символы с указанной по указанную позицию;
deleteCharAt(int index)	удалить символ в указанной позиции;
getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	сохранить последовательность символов в массив;

Работа со строками

indexOf(String str)	поиск первого вхождения подстроки;
indexOf(String str, int fromIndex)	поиск первого вхождения подстроки с указанной позиции;
lastIndexOf(String str)	поиск последнего вхождения подстроки;
lastIndexOf(String str, int fromIndex)	поиск последнего вхождения подстроки с указанной позиции;
replace(int start, int end, String str)	замена участка строки указанной строкой;
reverse()	расположить символы в обратном порядке;

Работа со строками

setCharAt(int index, char ch)	заменить символ в указанной позиции;
setLength(int newLength)	установить новый размер строки;
substring(int start)	вернуть подстроку с указанной позиции и до конца как строку;
substring(int start, int end)	вернуть подстроку как строку.

charAt(int index)	символ в указанной позиции;
length()	размер строки;
subSequence(int start, int end)	вернуть подстроку как последовательность символов;
toString()	вернуть строковое представление объекта.

Работа со строками

```
 //Неэффективно! Создаётся 99 новых объектов String
String output = "Some text";
int count = 100;
for (int i = 0; i < 100; i++) {
    output += i;
}

//Эффективно. Работаем с одним объектом
// StringBuilder outputSB = new StringBuilder(110);
StringBuffer outputSB = new StringBuffer(110);
outputSB.append("Some text");
for (int i = 0; i < count; i++) {
    outputSB.append(i);
}
```

Практика

1. Задайте строку. Заменить в заданной строке все пробелы знаками подчеркивания.
2. Дана строка. Определить длину строки в символах и в словах (разделителем между словами считать знак пробела). Вывести заданную строку в обратном порядке по символам и по словам.
3. Дан массив текстовых значений. Найти самый длинный элемент массива. Создать предложение из входящих в массив строк, самый длинный элемент массива разместить в начале предложения.
4. Дана строка. Подсчитать общее количество содержащихся в ней строчных латинских и русских букв.
5. Даны целые положительные числа N1 и N2 и строки S1 и S2. Получить из этих строк новую строку, содержащую первые N1 символов строки S1 и последние N2 символов строки S2 (в указанном порядке).
6. Даны строки S и S0. Удалить из строки S все подстроки, совпадающие с S0. Если совпадающих подстрок нет, то вывести строку S без изменений

Практика

7. Дана строка, содержащая по крайней мере один символ пробела. Вывести подстроку, расположенную между первым и последним пробелом исходной строки. Если строка содержит только один пробел, то вывести пустую строку
8. Дана строка, состоящая из слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова, разделенные одним символом «.» (точка). В конце строки точку не ставить.
9. Дана строка-предложение. Вывести самое короткое слово в предложении. Если таких слов несколько, то вывести последнее из них. Словом считать набор символов, не содержащий пробелов, знаков препинания и ограниченный пробелами, знаками препинания или началом/концом строки.

Практика

10. Дана строка-предложение на русском языке. Зашифровать её, выполнив циклическую замену каждой буквы на следующую за ней в алфавите и сохранив при этом регистр букв («А» перейдет в «Б», «а» — в «б», «Б»— в «В», «я» — в «а» и т. д.). Знаки препинания и пробелы не изменять.
11. Запишите десятичное число (от 1 до 3999) римскими цифрами (I,V,X,L,C,D,M – 1,5,10,50,100,500,1000).