Программирование на Java
Тема 2.2 Классы и объекты

Работа со строками

- **Пример:**

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

- **Компилятор выполнит:**

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

Работа со строками

- Очень большое место в обработке информации занимает работа с текстами. Текстовые строки в языке Java являются объектами. Они представляются экземплярами класса **String** или класса **StringBuffer**.
- В объектах класса **String** хранятся *строки-константы* неизменной длины и содержания. 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** (см. пример на слайде).
- Будет создан объект класса **SringBuffer**, в него последовательно добавлены строки "Это", " одна ", "строка", и получившийся объект класса **StringBuffer** будет приведен к типу **String** методом toString().
- Символы в строках хранятся в кодировке Unicode, в которой каждый символ занимает два байта. Тип каждого символа char.

Способы создания строк класса String

- **Создание ссылки типа String на строку-константу:**

```
String s1 = "Это строка.";
String s2 = "Это длинная строка, " +
"записанная в двух строках исходного текста";
String s = "";    //пустая строка
String s = null; //пустая ссылка
```

Класс String

- Перед работой со строкой ее следует создать. Способы создания строк представлены на слайде.
- Самый простой способ создать строку — это организовать ссылку типа string на строку-константу.
- Если константа длинная, можно записать ее в нескольких строках текстового редактора, связывая их операцией сцепления.
- **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.

Способы создания строк класса String

- **Создание объекта класса String с помощью операции new:**

```
String s3 = new String(); //объект инициализируется ""
```

Конструкторы класса String

- `String();`
- `String(String str);`
- `String(StringBuffer str);`
- `String(byte[] byteArray);`
- `String (char[] charArray);`
- `String (byte[] byteArray, int offset, int count);`
- `String (char[] charArray, int offset, int count);`
- `String(byte[] byteArray, String encoding;`
- `String(byte[] byteArray, int offset, int count, String encoding.`

Класс String

- Самый правильный способ создать объект с точки зрения ООП — создать объект класса String с помощью операции new вызвать его конструктор в операции new. Класс string предоставляет девять конструкторов (см. слайд):
  - **`String()`** — объект инициализируется пустой строкой;
  - **`String (String str)`** — объект инициализируется другим объектом класса String ;
  - **`String (StringBuffer str)`** — объект инициализируется другим объектом класса StringBuffer;
  - **`String(byte[] byteArray)`** — объект инициализируется 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.

Способы создания строк класса String

- **Создание строк с использованием статических методов:**

```
copyValueOf(char[] charArray)
copyValueOf(char[] charArray, int offset, int length)
```

- **Примеры:**

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

Класс String

- Методы copyValueOf(char[] charArray) и copyValueOf(char[] charArray, int offset, int length) создают строку по заданному массиву символов и возвращают ее в качестве результата своей работы. (примеры см. слайд). Получим в объекте s1 строку "Символьный", а в объекте s2 — строку "вольный".

Операции со строками

Сцепление строк

- **Примеры:**

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

attention += s;

System.out.println("2" + 2 + 2);
System.out.println(2 + 2 + "2");
System.out.println("2" + (2 + 2));
```

Определение длины строки

- **Примеры:**

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

или

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

Операции со строками

- Со строками можно производить операцию *сцепления строк* (concatenation), обозначаемую знаком плюс +. Эта операция создает новую строку, просто составленную из состыкованных первой и второй строк. Ее можно применять и к константам, и к переменным. Вторая операция — присваивание += — применяется к переменным в левой части (Примеры см. слайд).
- 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".

- Второй вариант определения длины строки допустим, поскольку строка-константа — полноценный объект класса string. Заметьте, что строка — это не массив, у нее нет поля length. Действительно, символы хранятся в массиве, но он закрыт, как и все поля класса string.

Выбор символа из строки

charAt(int ind)- выбирает символ с индексом ind

toCharArray() - возвращает массив символов из строки

```
getChars(int begin, int end, char[] dst, int ind)
getBytes()
getBytes(String encoding)-
```

Операции со строками

- 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.
- 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 ind + (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.
- 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).

Выбор подстроки из строки

```
substring(int begin, int end)
substring (int begin)
```

Примеры:

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

Операции со строками

- 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.
- Примеры на слайде: получим в строке sub1 значение "once", а в sub2 — значение "anywhere".

Сравнение строк

- Операция сравнения (==) сравнивает ссылки на строки
- Примеры:

```
String s1 = "Какая-то строка";
String s2 = "Другая строка";
s1 == s2 -> false
s1 = s2; // ссылки указывают на одну строку
s1 == s2 -> true
String s2 == "Какая-то строка";
s1 == s2 -> true // компилятор создаст только один
 // экземпляр константы "Какая-то строка" и
 // направит на него все ссылки.
```

Сравнение строк

- **Методы сравнения строк :**
  - `equals (object obj)`
  - `equalsIgnoreCase(object obj)`
  - `compareTo(string str)`
  - `compareToIgnoreCase(string str)`
  - `compareTo(Object obj)`
  - `regionMatches(int indl, String str, int ind2, int len)`
  - `regionMatches(boolean flag, int indl, String str, int ind2, int len)`

- **Примеры:**

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

s2.equals("другая строка") -> false
s2.equalsIgnoreCase("другая строка") -> true
```

Операции со строками

- 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.
- Метод **compareTo(string str)** возвращает целое число типа int, вычисленное по следующим правилам:
  - 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.
  - В первом случае возвращается значение this.charAt(k) - str.charAt(k), т. е. разность кодировок Unicode первых несовпадающих символов.
  - Во втором случае возвращается значение this.length() - str.length(), т. е. разность длин строк.
  - Если строки совпадают, возвращается о (нуль).
- 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.
- 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 '\u0401', а строчная буква е — после всех русских букв, ее код '\u0451'.

Операции со строками

- Задать свое размещение букв можно 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.

Операции со строками

Поиск символа в строке

- `indexOf(int ch)`
- `indexOf(int ch, int ind)`
- `lastIndexof (int ch)`
- `lastIndexof(int ch, int ind)`

- **Примеры:**

```
String s ="Молоко", s.indexOf('o') -> результат
                                      индекс 1
"молоко".indexof('o', indexof ('o') + 1)  -> 3.
"Молоко".lastindexof('o') -> 5
```

Операции со строками

- 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.
- 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 < 0, то поиск идет с начала строки, если ind больше длины строки, то символ не ищется, т. е. возвращается -1.
- 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.
- 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 < 0, то возвращается -1.

Операции со строками

Поиск подстроки

- `indexof (string sub)`
- `indexOf (string sub, int ind)`
- `lastindexof (string sub)`
- `lastIndexof(String stf, int ind)`
- `startsWith(string sub)`
- `startsWith(String sub, int ind)`
- `endsWitht(String sub)`

- **Примеры:**

```
"Раскраска".indexof ("рас") -> результат индекс 4
 if (fileName.endsWith(". Java"))
```

Операции со строками

- 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.
- Если вы хотите начать поиск не с начала строки, а с какого-то индекса ind, используйте метод **indexOf (string sub, int ind)**. если ind < 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 **lastIndexof(String stf, int ind)**. Если ind больше длины строки, то поиск идет от конца строки, если ind < 0, 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 **startsWith(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.
- Перечисленные выше методы создают исключительную ситуацию, если sub == null.
- Если вы хотите осуществить поиск, не учитывающий регистр букв, измените предварительно регистр всех символов строки.

Операции со строками

Смена регистра букв

- `toLowerCase()`
- `toUpperCase ()`
- `toLowerCase(Locale loc)` и `toUpperCase(Locale loc)`

Замена отдельного символа

- `replace (int old, int new)`
- **Примеры:**

```
"Рука в руку сует хлеб"
replace ('у', 'е') -> "Река в реке сеет хлеб"
```

Удаление пробелов в начале и конце строки

- `trim()`

Операции со строками

- 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 loc)** и **toUpperCase(Locale loc).**
- 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.
- Регистр букв при замене учитывается.
- Метод **trim()** возвращает новую строку, в которой удалены начальные и конечные символы с кодами, не превышающими '\u0020'.

Преобразование данных другого типа в строку

- `valueof (type elem)` – для `boolean`, `char`, `int`, `long`, `float`, `double`, массива `char[]`, и просто объекта типа `object`.
- `valueof(char[] ch, int offset, int len)`
- `toString()`
- `"" + elem -> elem.toString()`

Операции со строками

- 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** 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().**

Операции со строками

- **Объекты класса StringBuffer — это строки переменной длины**

- `ensureCapacity(int minCapacity)`
- `setLength(int newLength)`
- `length()`
- `capacity()`

Конструкторы класса StringBuffer

- `StringBuffer()`
- `StringBuffer (int capacity)`
- `StringBuffer (String str)`

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.
- **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 stringBuffer можно только с помощью операции new и конструкторов.
- 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.

Добавление подстроки

- `append (string str)`
- `append (type elem)`
- `append (char[] str)`
- `append (char[], sub, int offset, int len)`
- `append (Object obj)`

Вставка подстроки

- `insert (int ind, string str)`
- **Пример:**
- ```
  String s = new stringBuffer("Это большая строка")
  insert(4, "не").toString();
  s == "Это небольшая строка"
  ```
- `insert(int ind, type elem)`
- `insert(int ind, char[] str)`
- `insert(int ind, char[] sub, int offset, int len)`
- `insert(int ind, Object obj)`
  ```

Класс StringBuffer

- 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 ().**
- 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 на ту же преобразованную строку.
- Основной метод **insert (int ind, string str)** вставляет строку str в данную строку перед ее символом с индексом ind. Если ссылка str == null вставляется строка "null" (см. пример на слайде).
- 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:
- Десятый метод вставляет просто объект**:**
- Объект obj перед добавлением преобразуется в строку своим методом toString().

Удаление подстроки

- `delete (int begin, int end)`
- **Пример:**

```
String  s = new stringBuffer("Это небольшая строка")
delete(4, 6).toString(); -> s == "Это большая строка"
```

Удаление символа

- `deleteCharAt (int ind)`

Замена подстроки

- `replace (int begin, int end, string str)`

Переворот строки

- **Пример:**

```
String s = new stringBuffer("Это небольшая строка")
reverse().toString();
получим s == "акортс яашьлобен отЭ"
```

Класс StringBuffer

- Метод **delete (int 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 (пример см. на слайде). Если begin отрицательно, больше длины строки или больше end, возникает исключительная ситуация. Если begin == end, удаление не происходит.
- Метод **deleteCharAt (int ind)** удаляет символ с указанным индексом ind. Длина строки уменьшается на единицу. Если индекс ind отрицателен или больше длины строки, возникает исключительная ситуация.
- 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** ().
- Метод **reverse** () меняет порядок расположения символов в строке на обратный порядок.

Синтаксический разбор строки

Конструкторы класса StringTokenizer

- `StringTokenizer (string str)`
- `StringTokenizer (string str, string delimeters)`
- **Пример:** `StringTokenizer("Казнить,нельзя:пробелов-нет", " \t\n\r,:-");`
- `StringTokenizer(string str, string deiimeters, boolean flag);`

Методы класса StringTokenizer

- `nextToken()`
- `hasMoreTokens()`
- `countTokens ()`
- `nextToken(string newDeiimeters)`

Класс StringBuffer

- В пакет **java.util** входит простой класс StringTokenizer, облегчающий разбор строк.
- Класс StringTokenizer 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 delimeters.
- В примере первый разделитель — пробел. Потом идут символ табуляции, символ перевода строки, символ возврата каретки, запятая, двоеточие, дефис. Порядок расположения разделителей в строке delimeters не имеет значения. Разделители не включаются в число слов.
- Третий конструктор позволяет включить разделители в число слов. Если параметр flag равен true, то разделители включаются в число слов, если false — нет.
- 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.
- Метод **countTokens ()** возвращает число оставшихся слов.
- Метод **nextToken(string newDelimeters)** позволяет "на ходу" менять разделители. Следующее слово будет выделено по новым разделителям newDelimeters; новые разделители действуют далее вместо старых разделителей, определенных в конструкторе или предыдущем методе nextToken ().

Пример Найти количество вхождений заданного слова в текст:

```
import java.lang.*;
import java.io.*;
import java.util.*;

public class Main
{
   public static void main(String[] args) throws
  Exception
   {
     String str, word, si;
     int k = 0;
     Scanner in = new Scanner(System.in);
     System.out.println("Введите слово:");
     word = in.nextLine();

     BufferedReader r = new BufferedReader(
                                new FileReader("info.txt"));
```

Строки

```
while ((str = r.readLine()) != null)
 {  System.out.println(str);
    StringTokenizer st = new StringTokenizer(str, "
  \t\n\r,.-");
      while(st.hasMoreTokens())
       { si = st.nextToken(); // Получаем слово
      if (word.equalsIgnoreCase(si))
          k++;
       }
 }
  System.out.println("Количество вхождений  слова = " + k);
   }
}
```

```
Введите слово:
string1
String1 string2, string1. String3 string1.
String4 - string1.
Количество вхождений  слова = 4
```

Контрольные вопросы

1. Сроки класса String: способы создания, примеры использования основных методов.
2. Сроки класса StringBuffer : способы создания, примеры использования основных методов.