1. **Какие примитивные типы есть в Java**

boolean 32 - булев тип, может иметь значения true или false

byte - 8-разрядное целое число

short - 16-разрядное целое число

int - 32-разрядное целое число

long - 64-разрядное целое число

char - 16-разрядное беззнаковое целое, представляющее собой символ UTF-16 (буквы и цифры)

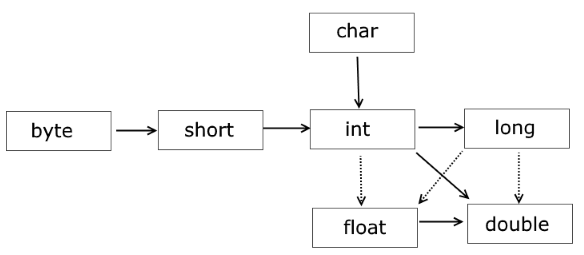
float - 32-разрядное число в формате IEEE 754 с плавающей точкой

double - 64-разрядное число в формате IEEE 754 с плавающей точкой

1. **Что такое явные и неявные приведения, с чем связано их наличие?**

Когда в одной операции вовлечены данные разных типов, не всегда необходимо использовать операцию преобразования типов. Некоторые виды преобразований выполняются неявно, автоматически.

**Автоматические преобразования**



Стрелками на рисунке показано, какие преобразования типов могут выполняться автоматически. Пунктирными стрелками показаны автоматические преобразования с потерей точности.

Автоматически без каких-либо проблем производятся расширяющие преобразования (widening) - они расширяют представление объекта в памяти.

**Автоматические преобразования с потерей точности**

Некоторые преобразования могут производиться автоматически между типами данных одинаковой разрядности или даже от типа данных с большей разрядностью к типа с меньшей разрядностью. Это следующие цепочки преобразований: int -> float, long -> float и long -> double. Они производятся без ошибок, но при преобразовании мы можем столкнуться с потерей информации.

Явные преобразования

Во всех остальных преобразованиях примитивных типов явным образом применяется операция преобразования типов. Обычно это сужающие преобразования (narrowing) от типа с большей разрядностью к типу с меньшей разрядностью

1. **Какие данные мы рискуем потерять при явных приведениях?**

Потеря данных при преобразовании

При применении явных преобразований мы можем столкнуться с потерей данных. Например, в следующем коде у нас не возникнет никаких проблем:

|  |  |
| --- | --- |
| 1  2  3 | int a = 5;  byte b = (byte) a;  System.out.println(b);      // 5 |

Число 5 вполне укладывается в диапазон значений типа byte, поэтому после преобразования переменная b будет равна 5. Но что будет в следующем случае:

|  |  |
| --- | --- |
| 1  2  3 | int a = 258;  byte b = (byte) a;  System.out.println(b);      // 2 |

Результатом будет число 2. В данном случае число 258 вне диапазона для типа byte (от -128 до 127), поэтому произойдет усечение значения. Почему результатом будет именно число 2?

Число a, которое равно 258, в двоичном системе будет равно 00000000 00000000 00000001 00000010. Значения типа byte занимают в памяти только 8 бит. Поэтому двоичное представление числа int усекается до 8 правых разрядов, то есть 00000010, что в десятичной системе дает число 2.

Усечение рациональных чисел до целых

При преобразовании значений с плавающей точкой к целочисленным значениям, происходит усечение дробной части:

|  |  |
| --- | --- |
| 1  2 | double a = 56.9898;  int b = (int)a; |

Здесь значение числа b будет равно 56, несмотря на то, что число 57 было бы ближе к 56.9898. Чтобы избежать подобных казусов, надо применять функцию округления, которая есть в математической библиотеке Java:

|  |  |
| --- | --- |
| 1  2 | double a = 56.9898;  int b = (int)Math.round(a); |

1. **StringBuilder.**

Объекты String являются неизменяемыми, поэтому все операции, которые изменяют строки, фактически приводят к созданию новой строки, что сказывается на производительности приложения. Для решения этой проблемы, чтобы работа со строками проходила с меньшими издержками в Java были добавлены классы StringBuffer и StringBuilder. По сути они напоминает расширяемую строку, которую можно изменять без ущерба для производительности.

Эти классы похожи, практически двойники, они имеют одинаковые конструкторы, одни и те же методы, которые одинаково используются. Единственное их различие состоит в том, что класс StringBuffer синхронизированный и потокобезопасный. То есть класс StringBuffer удобнее использовать в многопоточных приложениях, где объект данного класса может меняться в различных потоках. Если же речь о многопоточных приложениях не идет, то лучше использовать класс StringBuilder, который не потокобезопасный, но при этом работает быстрее, чем StringBuffer в однопоточных приложениях.

StringBuffer определяет четыре конструктора:

Аналогичные конструкторы определяет StringBuilder:

|  |  |
| --- | --- |
| 1  2  3  4 | StringBuilder()  StringBuilder(int capacity)  StringBuilder(String str)  StringBuilder(CharSequence chars) |

Рассмотрим работу этих классов на примере функциональности StringBuffer.

При всех операциях со строками StringBuffer / StringBuilder перераспределяет выделенную память. И чтобы избежать слишком частого перераспределения памяти, StringBuffer/StringBuilder заранее резервирует некоторую область памяти, которая может использоваться. Конструктор без параметров резервирует в памяти место для 16 символов. Если мы хотим, чтобы количество символов было иным, то мы можем применить второй конструктор, который в качестве параметра принимает количество символов.

Третий и четвертый конструкторы обоих классов принимают строку и набор символов, при этом резервируя память для дополнительных 16 символов.

С помощью метода capacity() мы можем получить количество символов, для которых зарезервирована память. А с помощью метода ensureCapacity() изменить минимальную емкость буфера символов:

Метод charAt() получает, а метод setCharAt() устанавливает символ по определенному индексу:

Метод getChars() получает набор символов между определенными индексами:

Метод append() добавляет подстроку в конец StringBuffer:

Метод insert() добавляет строку или символ по определенному индексу в StringBuffer:

Метод delete() удаляет все символы с определенного индекса о определенной позиции, а метод deleteCharAt() удаляет один символ по определенному индексу:

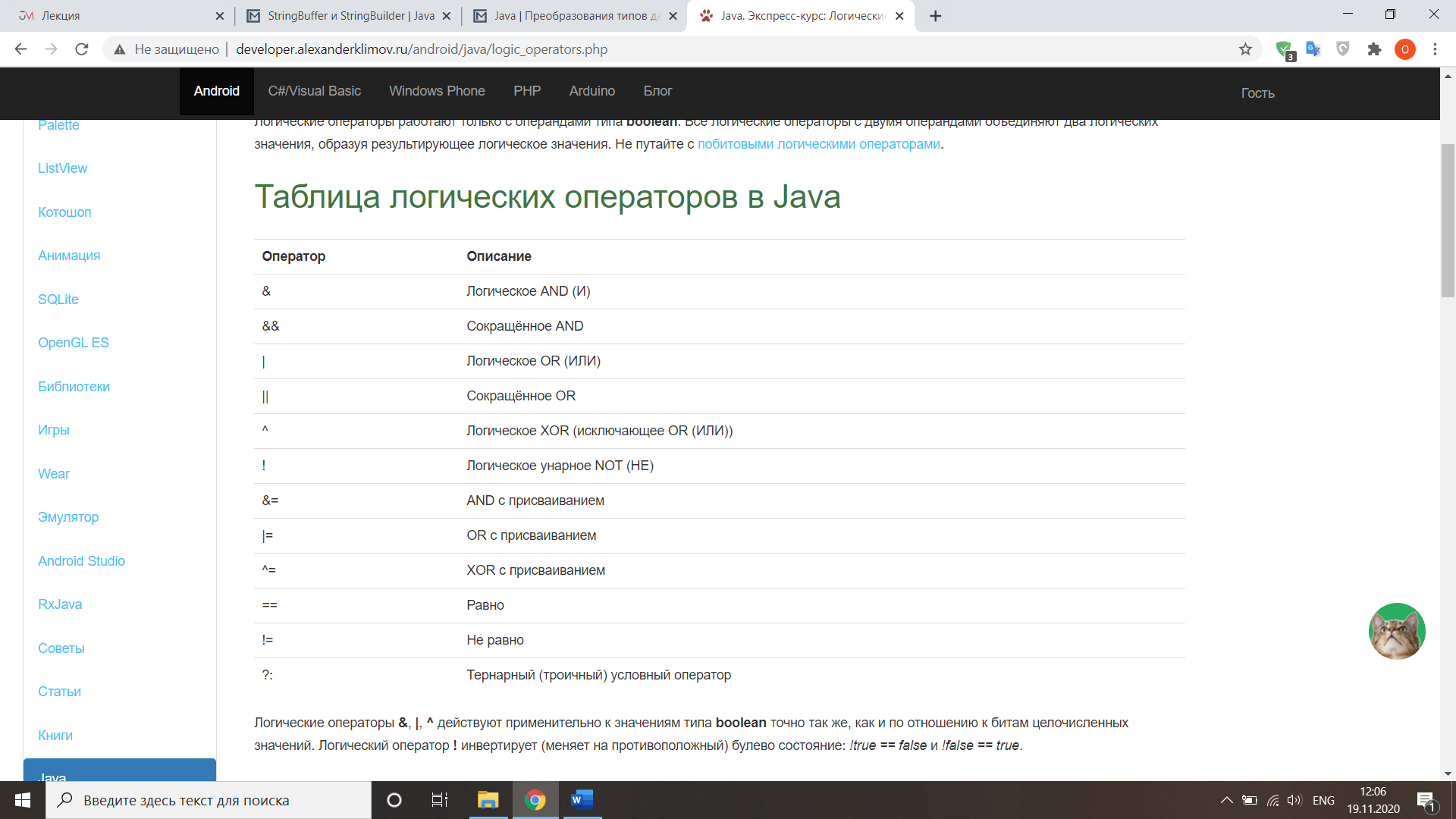
Метод substring() обрезает строку с определенного индекса до конца, либо до определенного индекса:

Для изменения длины StringBuffer (не емкости буфера символов) применяется метод setLength(). Если StringBuffer увеличивается, то его строка просто дополняется в конце пустыми символами, если уменьшается - то строка по сути обрезается:

Для замены подстроки между определенными позициями в StringBuffer на другую подстроку применяется метод replace():

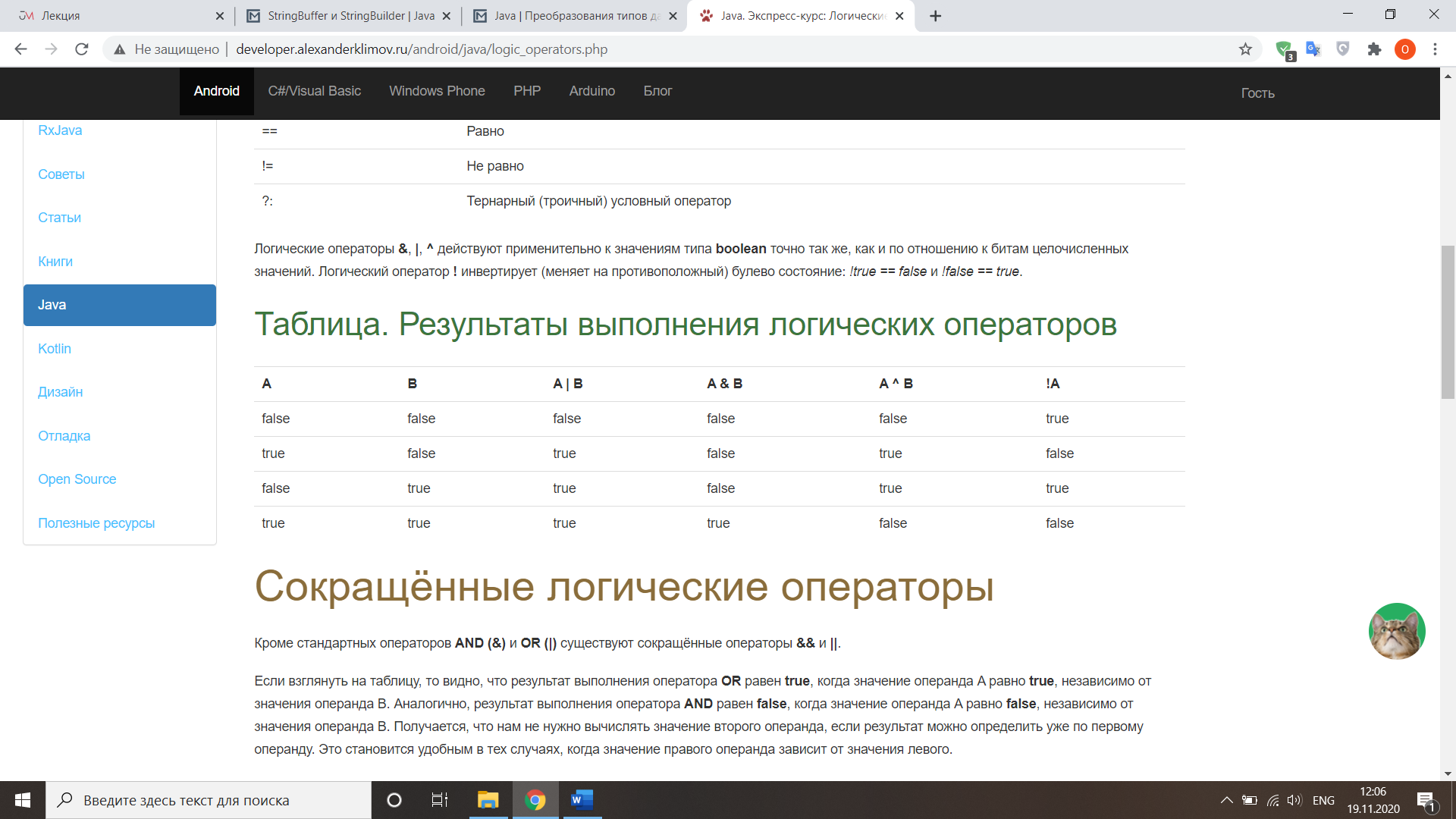
Метод reverse() меняет порядок в StringBuffer на обратный:

1. **Логические операторы.**



Логические операторы &, |, ^ действуют применительно к значениям типа boolean точно так же, как и по отношению к битам целочисленных значений. Логический оператор ! инвертирует (меняет на противоположный) булево состояние: !true == false и !false == true.

Таблица. Результаты выполнения логических операторов



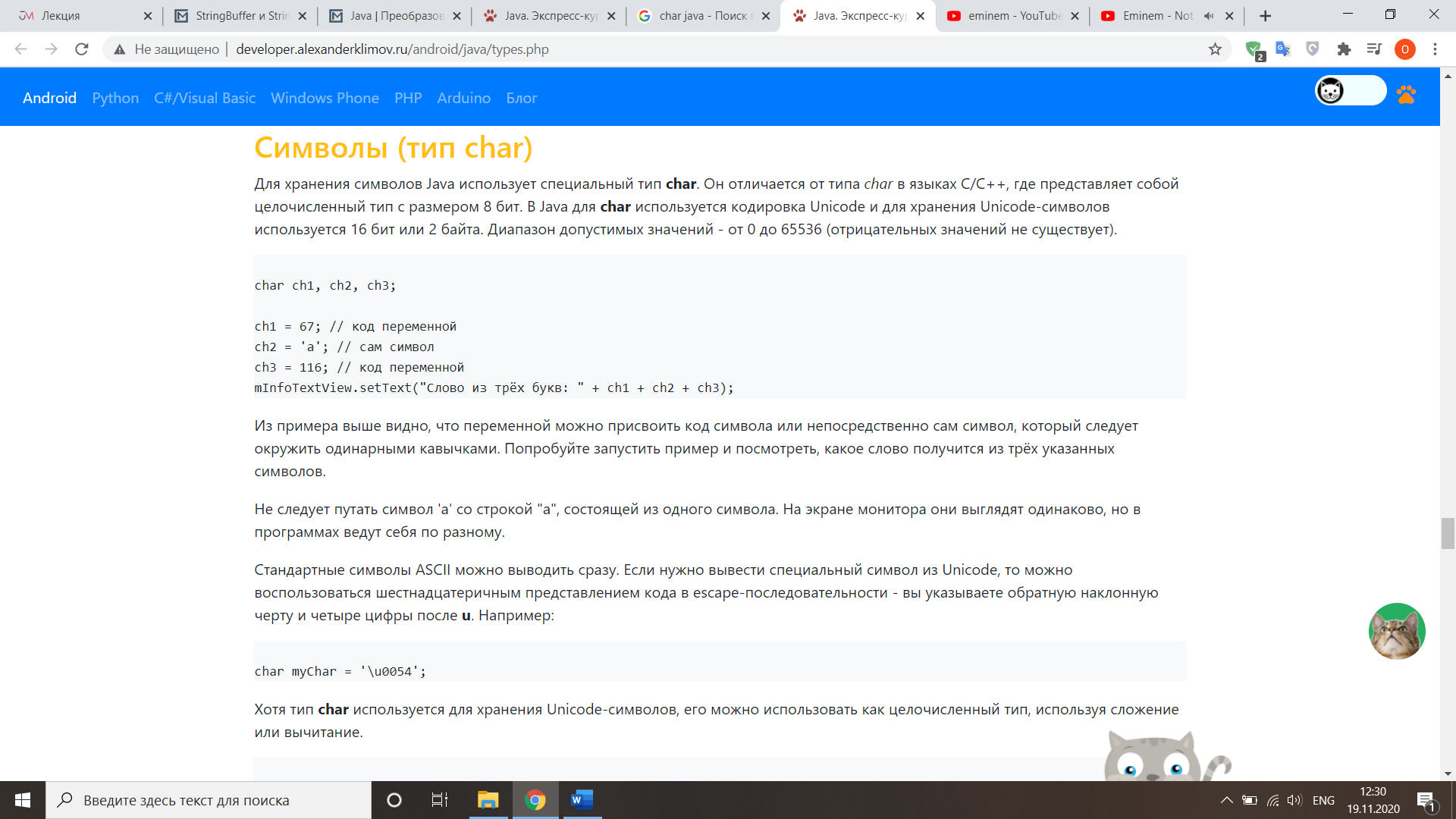
**Сокращённые логические операторы**

Кроме стандартных операторов AND (&) и OR (|) существуют сокращённые операторы && и ||.

Если взглянуть на таблицу, то видно, что результат выполнения оператора OR равен true, когда значение операнда A равно true, независимо от значения операнда B. Аналогично, результат выполнения оператора AND равен false, когда значение операнда A равно false, независимо от значения операнда B. Получается, что нам не нужно вычислять значение второго операнда, если результат можно определить уже по первому операнду. Это становится удобным в тех случаях, когда значение правого операнда зависит от значения левого.

1. **char**

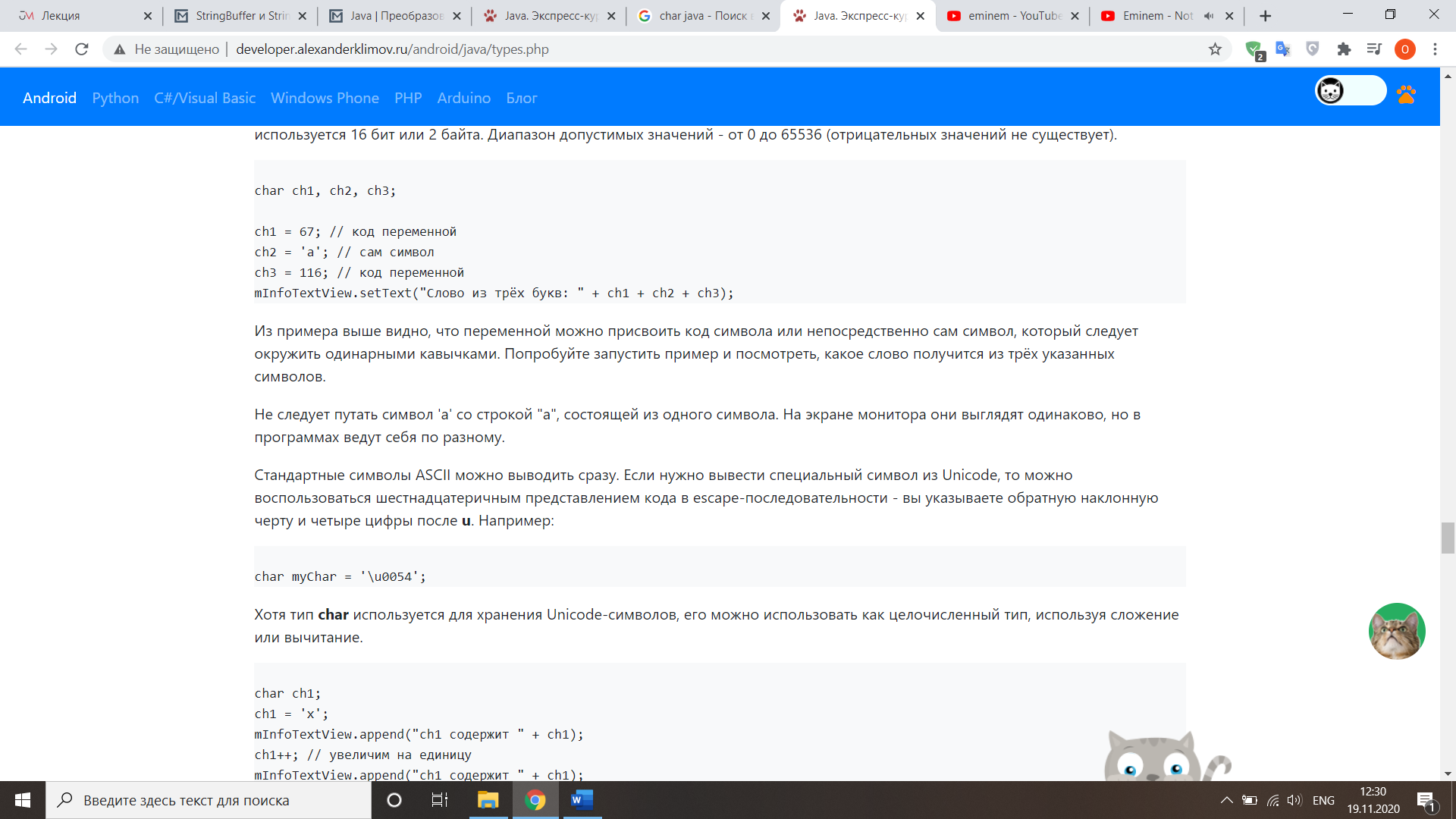
Для хранения символов Java использует специальный тип char. Он отличается от типа char в языках C/C++, где представляет собой целочисленный тип с размером 8 бит. В Java для char используется кодировка Unicode и для хранения Unicode-символов используется 16 бит или 2 байта. Диапазон допустимых значений - от 0 до 65536 (отрицательных значений не существует).



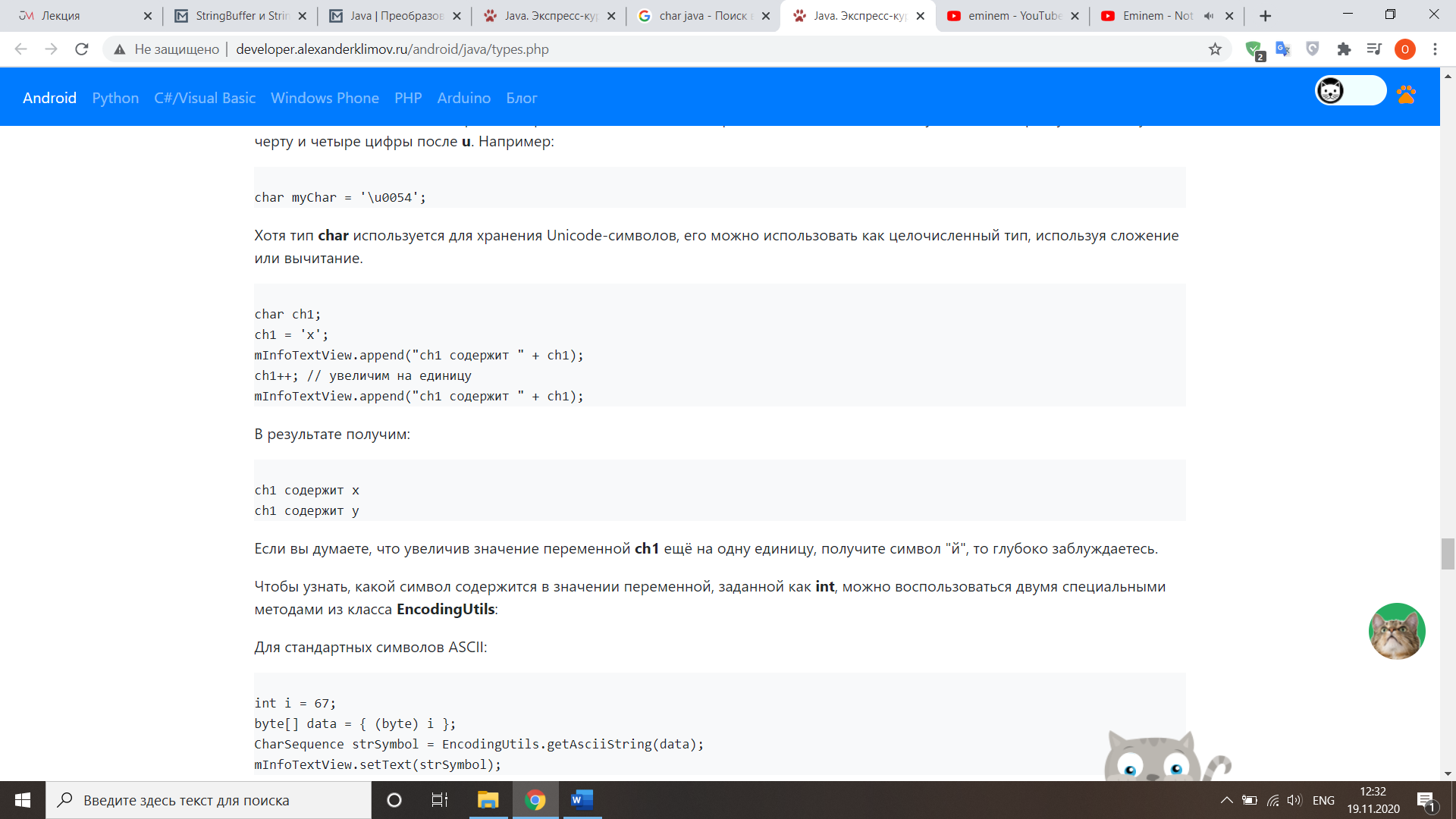
Из примера выше видно, что переменной можно присвоить код символа или непосредственно сам символ, который следует окружить одинарными кавычками. Попробуйте запустить пример и посмотреть, какое слово получится из трёх указанных символов.

Не следует путать символ 'a' со строкой "a", состоящей из одного символа. На экране монитора они выглядят одинаково, но в программах ведут себя по разному.

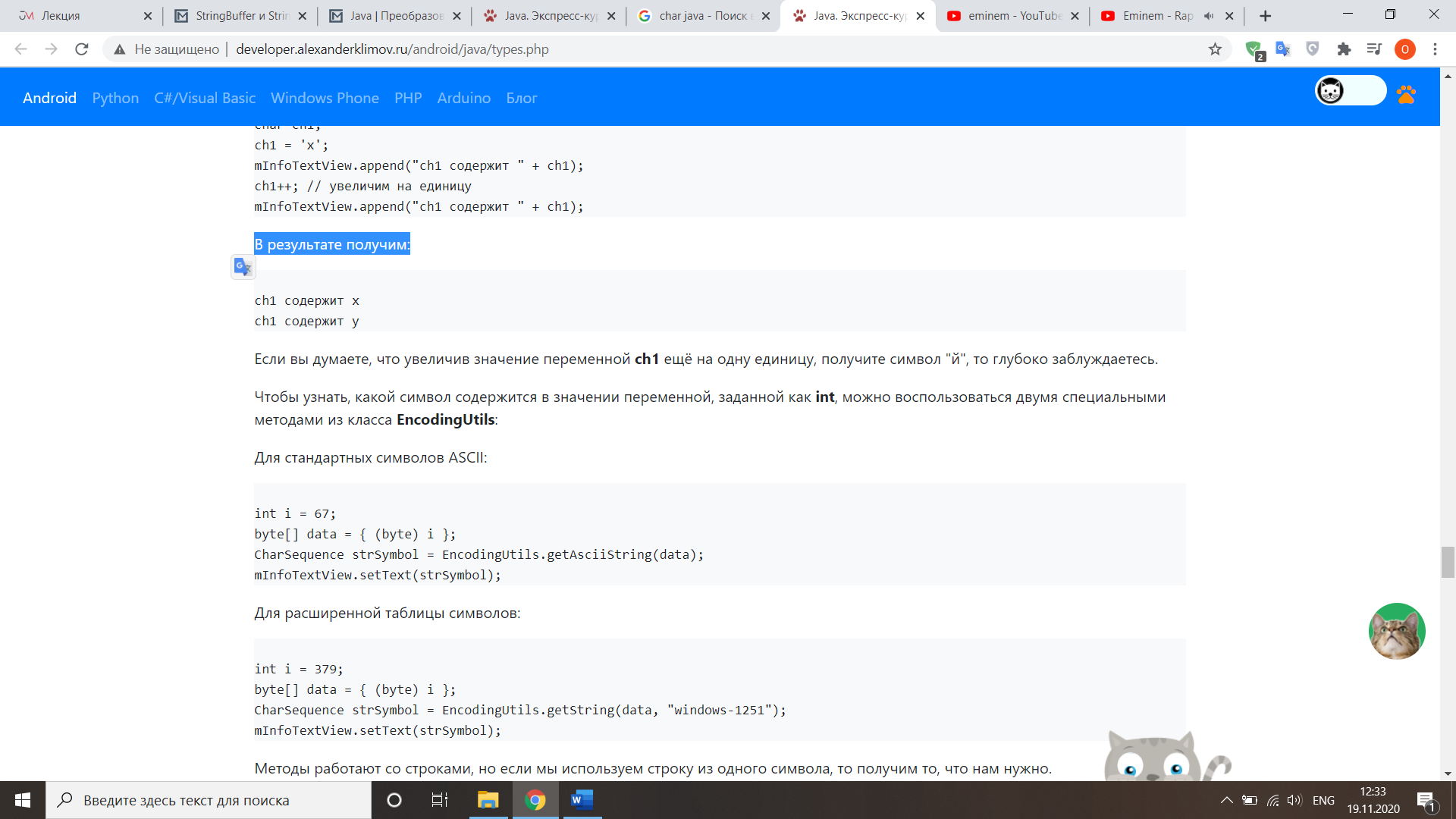
Стандартные символы ASCII можно выводить сразу. Если нужно вывести специальный символ из Unicode, то можно воспользоваться шестнадцатеричным представлением кода в escape-последовательности - вы указываете обратную наклонную черту и четыре цифры после u. Например:



Хотя тип char используется для хранения Unicode-символов, его можно использовать как целочисленный тип, используя сложение или вычитание.



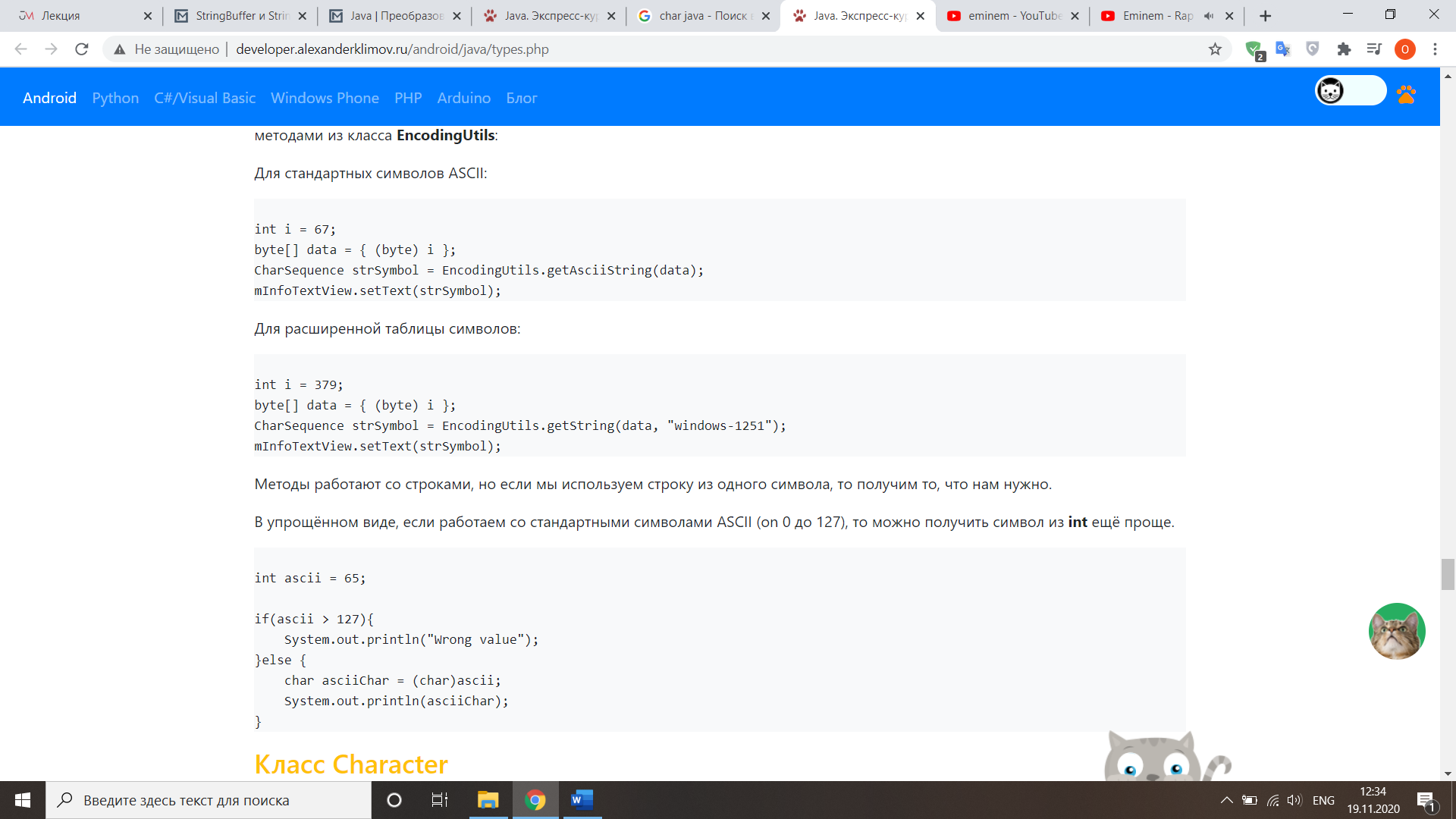
В результате получим:



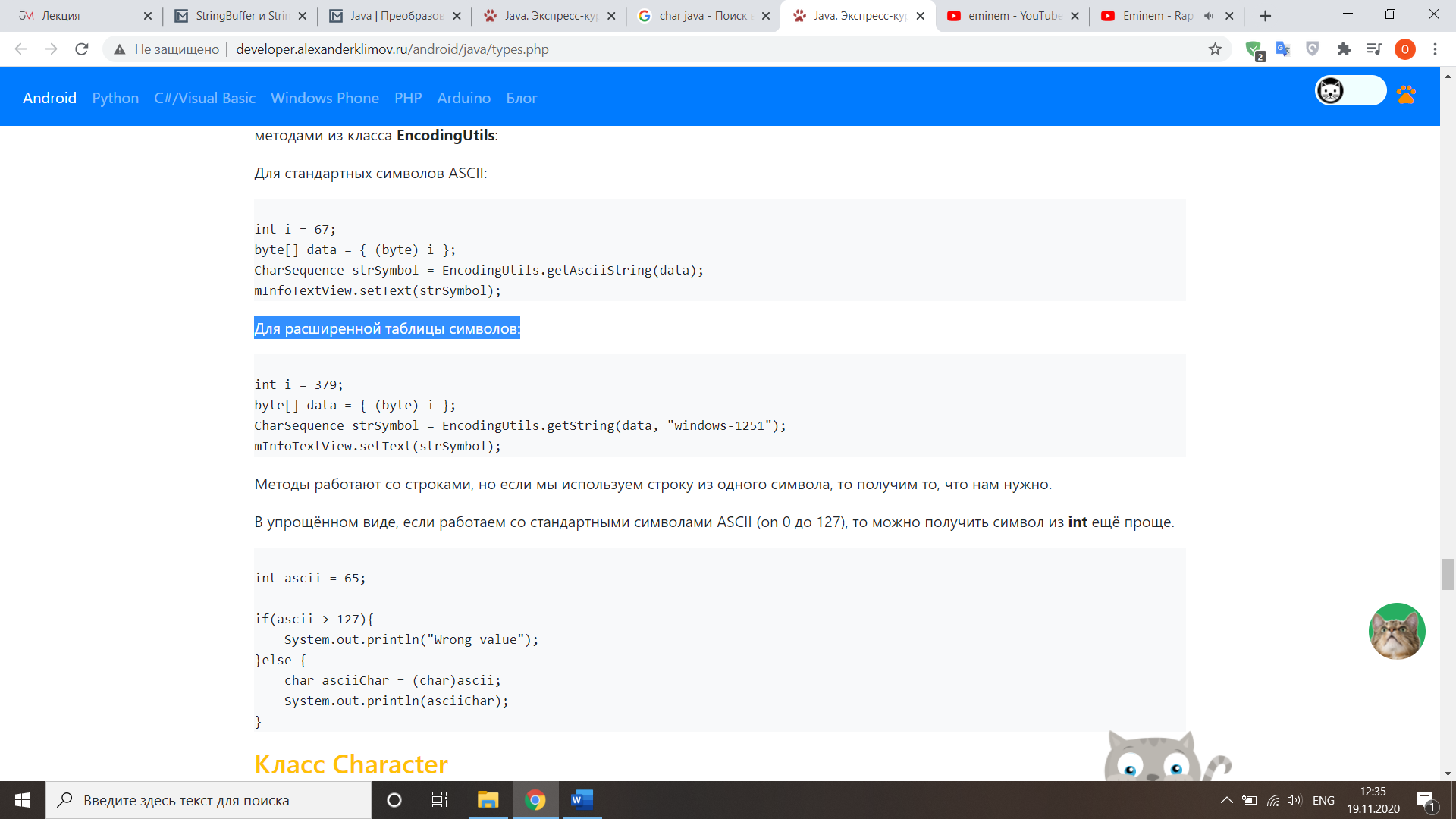
Если вы думаете, что увеличив значение переменной ch1 ещё на одну единицу, получите символ "й", то глубоко заблуждаетесь.

Чтобы узнать, какой символ содержится в значении переменной, заданной как int, можно воспользоваться двумя специальными методами из класса EncodingUtils:

Для стандартных символов ASCII:

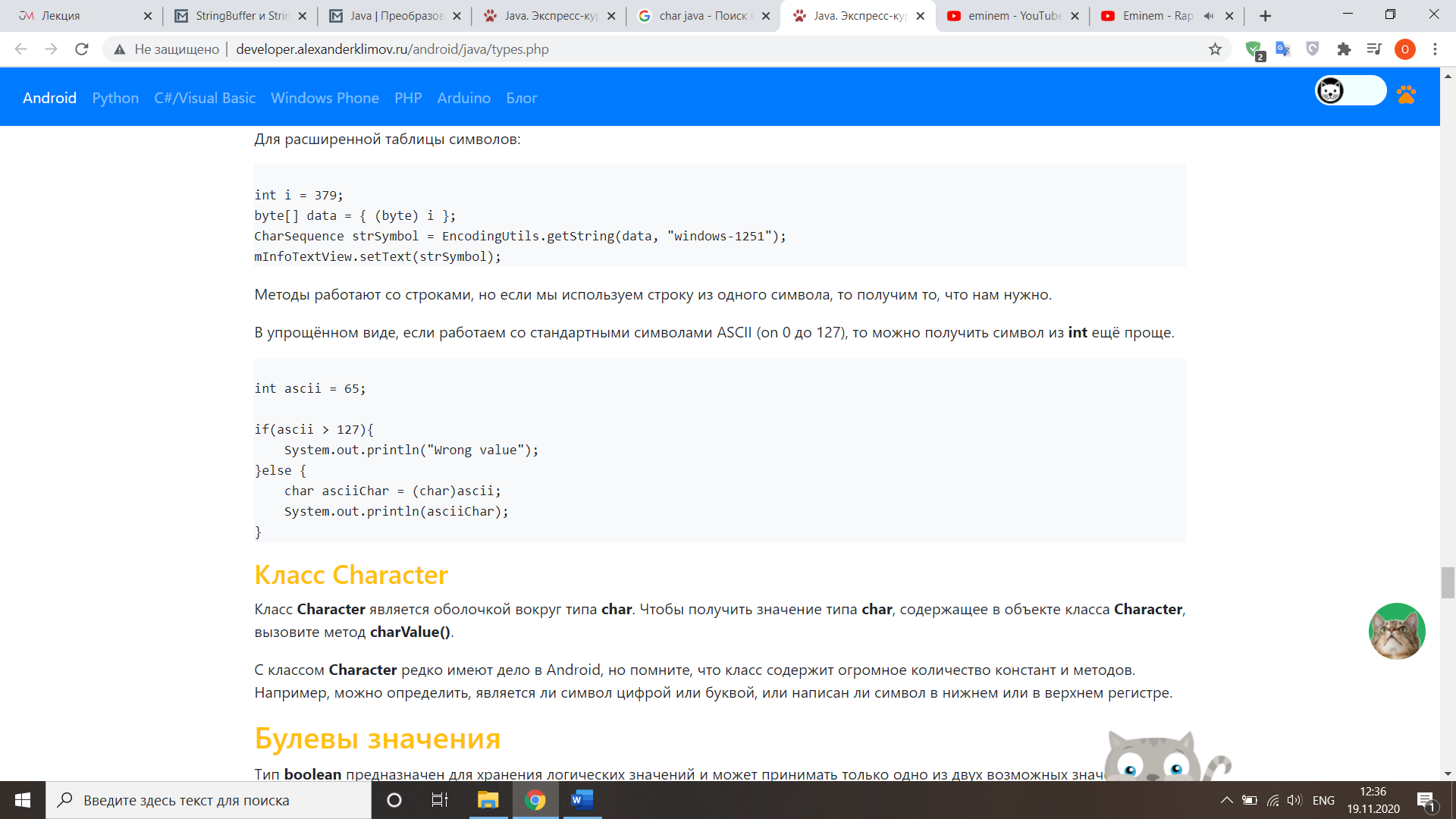


Для расширенной таблицы символов:



Методы работают со строками, но если мы используем строку из одного символа, то получим то, что нам нужно.

В упрощённом виде, если работаем со стандартными символами ASCII (on 0 до 127), то можно получить символ из int ещё проще.



1. **Неизменяемые типы**

Но в Java есть несколько типов данных, которые отличаются особым состоянием. Они являются неизменяемыми, или Immutable. Это значит, что если класс неизменяемый, состояние его объектов изменить невозможно.

Cамый известный пример Immutable - класса — String!

Неизменяемый класс – это класс, состояние которого не может быть изменено после создания. Здесь состоянием объекта по существу считаются значения, хранимые в экземпляре класса, будь то примитивные типы или ссылочные типы.

Для того чтобы сделать класс неизменяемым, необходимо выполнить следующие условия:

1. Не предоставляйте сеттеры или методы, которые изменяют поля или объекты, ссылающиеся на поля. Сеттеры подразумевают изменение состояния объекта а это то, чего мы хотим тут избежать.
2. Сделайте все поля final и private. Поля, обозначенные private, будут недоступными снаружи класса, а обозначение их final гарантирует, что вы не измените их даже случайно.
3. Не разрешайте субклассам переопределять методы. Самый простой способ это сделать – объявить класс как final. Финализированные классы в Java не могут быть переопределены.
4. Всегда помните, что ваши экземпляры переменных могут быть либо изменяемыми, либо неизменяемыми. Определите их и возвращайте новые объекты со скопированным содержимым для всех изменяемых объектов (ссылочные типы). Неизменяемые переменные (примитивные типы) могут быть безопасно возвращены без дополнительных усилий.
5. **Виды приведения. И правила приведения при наличии типов double float long int и тд.**
6. **Как добавить String’у pool строк?**

С помощью метода intern().

Когда метод intern() вызван, если пул строк уже содержит строку, эквивалентную к нашему объекту, что подтверждается методом equals(Object), тогда возвращается ссылка на строку из пула. В противном случае объект строки добавляется в пул и ссылка на этот объект возвращается. Этот метод всегда возвращает строку, которая имеет то же значение, что что и текущая строка, но гарантирует что это будет строка из пула уникальных строк.

1. **Расскажи про pool строк и pool примитивов**

Пул строк (String Pool) — это множество строк в кучи (Java Heap Memory). Мы знаем, что String — особый класс в java, с помощью которого мы можем создавать строковые объекты..

Сам строковый пул возможен только потому, что строки в Java неизменные. Также пул строк позволяет сохранить память в Java Runtime, хотя это и требует больше времени на создание самой строки.

Пример работы с пулом строк

Когда мы используем двойные кавычки, чтобы создать новую строку, то первым делом идет поиск строки с таким же значением в пуле строк. Если java такую строку нашла, то возвращает ссылку, в противном случае создается новая строка в пуле, а затем возвращается ссылка.

В Java есть пул(pool) целых чисел в промежутке [-128;127]. Т.е. если мы создаем Integer в этом промежутке, то вместо того, чтобы каждый раз создавать новый объект, JVM берет их из пула.

1. **Что такое инкремент и декремент?**

Инкремент (++) — это операция во многих языках программирования, которая выполняет увеличение переменной. Чаще всего под инкрементом подразумевается увеличение переменной на 1 единицу.

Декремент (--) — это уменьшение переменной. Чаще всего также на единицу.

Prefix, или Префиксная форма, описывается перед переменной "++x или --x", изначально икрементирует или декрементирует переменную.

Postfix, или Постфиксная форма, описывается после переменной "x++ или x--", икрементирует или декрементирует переменную после вычисления.

1. **Какая размерность у boolean?**

8 (при использовании в массивах), 32 (при использовании не в массивах)

1. **При сложении char + short какой результирующий тип получим?**

Int

1. **Расскажи про unboxing, autoboxing в обертках**

Автоупаковка

Это автоматическая инкапсуляция примитивного типа в эквивалентную ему класс-обёртку всякий раз, когда требуется объект данного типа. Про [инкапсуляцию и другие принципы ООП](https://habrahabr.ru/post/147927/) существует замечательная статья от [ffriend](https://habrahabr.ru/users/ffriend/).  
  
Autoboxing происходит:

* При присвоении значения примитивного типа переменной соответствующего класса-обёртки.
* При передаче примитивного типа в параметр метода, ожидающего соответствующий ему класс-обёртку.

Автораспаковка

Это преобразование класса-обёртки в соответствующий ему примитивный тип. Если при распаковке класс-обёртка был равен null, произойдет исключение java.lang.NullPointerException.  
   
Unboxing происходит:

* При присвоении экземпляра класса-обёртки переменной соответствующего примитивного типа.
* В выражениях, в которых один или оба аргумента являются экземплярами классов-обёрток (кроме операции == и !=).
* При передаче объекта класса-обёртки в метод, ожидающий соответствующий примитивный тип.

1. **Что такое массив и какие на нём есть ограничения?**

Массив — это структура данных, в которой хранятся элементы одного типа. Его можно представить, как набор пронумерованных ячеек, в каждую из которых можно поместить какие-то данные (один элемент данных в одну ячейку). Доступ к конкретной ячейке осуществляется через её номер. Номер элемента в массиве также называют индексом.

1. **Какая максимальная длина массива?**

Java использует целое число в качестве индекса массива, а максимальное целочисленное хранилище JVM составляет 2 ^ 32. так что вы можете хранить 2 147 483 647 элементов в массиве.

Если вам нужно больше, чем max-length, вы можете использовать два разных массива, но рекомендуемый метод - сохранить данные в файл. потому что хранение данных в файле не имеет ограничений. потому что файлы хранятся в ваших драйверах хранения, но массив хранятся в JVM. JVM предоставляет ограниченное пространство для выполнения программы.

1. **Расскажи про все условные операторы?**

If else, switch, тернарный оператор

1. **2 случая (правила/условия) в рекурсивном алгоритме?**

Базис рекурсии – условие выхода из блока рекурсивных вызовов – базисное решение задачи, при условиях, когда нет необходим ости вызывать рекурсию.

Шаг рекурсии – вызов функцией самой себя при изменении параметров.

1. **чем отличается метод от функции?**

ничем

Полезные ссылки:

Классы обертки

https://javarush.ru/groups/posts/1948-objertki-raspakovka-i-zapakovka

Что такое инкрементирование и декрементирование

<https://javarush.ru/groups/posts/1250-chto-takoe-inkrementirovanie-i-dekrementirovanie>

Побитовые операции

<https://javarush.ru/groups/posts/1925-pobitovihe-operacii>

Устройство вещественных чисел

https://javarush.ru/groups/posts/2136-ustroystvo-vejshestvennihkh-chisel