Лекция 7. Введение в язык программирования Java. Языковые лексемы Java. Введение в систему типов языка Java. Работа с примитивными типами и константами. Операции языка Java. Преобразование простых типов.

> Лексема - последовательность допустимых символов языка **программирования**, имеющая смысл для транслятора. Транслятор рассматривает программу как последовательность **лексем**.

Виды лексем

- идентификаторы (identifiers);
- ключевые слова (key words);
- литералы (literals);
- разделители (separators);
- операторы (operators).

Рассмотрим их по отдельности.

Идентификаторы

> Идентификаторы - это имена, которые даются различным элементам языка для упрощения доступа к ним.

Имена имеют пакеты, классы, интерфейсы, поля, методы, аргументы и локальные переменные.

Идентификаторы можно записывать символами ***Unicode***, то есть, на любом удобном языке.

Длина имени не ограничена.

Идентификатор состоит из **букв и цифр**.

- Имя не может начинаться с цифры.
- Java-буквы, используемые в идентификаторах, включают в себя **ASCII-символы A-Z (\u0041-\u005a), a-z (\u0061-\u007a)**, а также **знаки подчеркивания** _ (ASCII underscore, \u005f) и доллара $ (\u0024).
- **Знак доллара** используется только при **автоматической генерации кода** (чтобы исключить случайное совпадение имен), либо при использовании каких-либо старых библиотек, в которых допускались имена с этим символом.
- Java-цифры включают в себя обычные **ASCII-цифры 0-9** (\u0030-\u0039).

Для идентификаторов **не допускаются совпадения с зарезервированными словами** (это ключевые слова, булевские литералы **true** и **false** и **null**-литерал **null**). Конечно, если 2 идентификатора включают в себя разные буквы, которые одинаково выглядят (например, латинская и русская буквы A), то они считаются различными.

> Идентификаторы: Character, a, b, c, D, x1, x2, Math, sqrt, x, y, i, s, PI, getRadius, circle, getAbs, calculate, condition, getWidth, getHeight, java, lang, String

Разделители

( ) [ ] { } ; . ,

Операторы

Операторы используются в различных операциях - **арифметических**, **логических**, **битовых**, **операции сравнения, присваивания**. Следующие 37 лексем (все состоят только из ASCII-символов) являются операторами языка Java:

> = > < ! ~ ? : == <= >= != && || ++ -- + - * / & | ^ % << >> >>> += -= *= /= &= |= ^= %= <<= >>= >>>=

Ключевые слова

> abstract, default, if, private, this, boolean, do, implements, protected, throw, break, double, import, public, throws, byte, else, instanceof, return, transient, case, extends, int, short, try, catch, final, interface, static, void, char, finally, long, strictfp, volatile, class, float, native, super, while, const, for, new, switch, continue, goto, package, synchronized

Ключевые слова **goto** и **const** зарезервированы, но не используются. Это сделано для того, чтобы компилятор мог правильно отреагировать на использование этих распространенных в других языках слов. Напротив, оба булевских литерала **true, false** и **null**-литерал **null** часто считают ключевыми словами (возможно потому, что многие средства разработки подсвечивают их таким же образом), однако это именно литералы.

Введение в систему типов языка Java

Тип	Размер, байт	Диапазон значений	Значение по умолчанию	Описание
`byte`	1	-128 .. 127	`0`	Самое маленькое целое — один байт
`short`	2	-32,768 .. 32,767	`0`	Короткое целое, два байта

`int`	4	-2,147,483,648 до 2,147,483,647	`0`	Целое число, 4 байта
`long`	8	-9 223 372 036 854 775 807 до +9 223 372 036 854 775 807	`0L`	Длинное целое, 8 байт
`float`	4	-1038 .. 1038	`0.0f`	Дробное, 4 байта
`double`	8	-10308 .. 10308	`0.0d`	Дробное, двойной длины, 8 байт
`boolean`	1	`true`, `false`	`false`	Логический тип (только `true` & `false`)
`char`	2	0 .. 65,535	`'\u0000'`	Символы, 2 байта, все больше 0

> **Зеленые** - целочисленные, **синие** - дробные

Значение по умолчанию

> 💡 Кстати, **важный нюанс**. Если вы объявили переменную-класса (поле класса) или статическую переменную-класса, и не присвоили ей никакого значения, она инициализируется *значением по умолчанию*. Список таких значений приведен в таблице.

Локальные переменные методов не имеют значений по умолчанию, и если вы не присвоили такой переменной никакого значения, она считается не инициализированной и использовать ее значение нельзя.

Целые типы

В языке Java аж 4 целых типа: `byte`, `short`, `int` и `long`. Они отличаются размером и диапазоном значений, которые могут хранить.

Тип `int`

Самым часто используемым является тип `int`. Его название происходит от *Integer* (целый). Все целочисленные литералы в коде имеют тип `int` (если в конце числа не указана буква `L`, `F` или `D`).

Переменные этого типа могут принимать значение от `-2,147,483,648` до `+2,147,483,647`.

Это достаточно много и хватает почти для всех случаев жизни. Почти все функции, которые возвращают число, возвращают число типа `int`.

Примеры:

Код	Пояснение
`int n = "Строка".length();`	Метод `length()` возвращает длину строки
`String[] array = {"Tic", "Tac", "Toe"};` `int n = array.length;`	Поле `length` содержит длину массива.

Тип `short`

Тип `short` получил свое название от `short int`. Его еще называют *короткое целое*. В отличие от типа `int`, его длина всего два байта и возможный диапазон значений от `-32,768` до `+32,767`.

То есть в нем даже число миллион не сохранишь. Даже 50 тысяч. Это самый редко используемый целочисленный тип в Java. В основном его используют, когда хотят сэкономить на памяти.

Допустим, у вас ситуация, когда заранее известно, что значения с которыми вы работаете не превышает 30 тысяч, и таких значений миллионы.

Например, вы пишете приложение, которое обрабатывает картинки сверхвысокой четкости: на один цвет приходится `10` бит. А точек у вас в картинке — миллион. И вот тут уже играет роль, используете вы тип `int` или `short`.

Тип `long`

Этот тип получил свое название от `long int` — его еще называют *длинное целое*. В отличие от типа `int`, у него просто гигантский диапазон значений: от `-9*10 18` до `+9*10 18`

> **Почему же он не является основным целым типом?** Все дело в том, что Java появилась еще в середине 90-х, когда большинство компьютеров были 32-х разрядными. А это

> значило, что все процессоры были заточены под работу с числами из 32-х бит. С целыми числами из 64-х бит процессоры работать уже умели, но операции с ними были медленнее. Поэтому программисты разумно решили сделать стандартным целым типом тип `int`, ну а тип `long` использовать только тогда, когда без него действительно не обойтись.

Тип `byte`

Это самый маленький целочисленный тип в Java, но далеко не самый редко используемый. Его название совпадает со словом `byte` — минимальная адресуемая ячейка памяти в Java.

Размер допустимых значений типа `byte` не так уж велик: от `-128` до `+127`. Но не в этом его сила. Тип `byte` чаще всего используется, когда нужно хранить в памяти большой блок обезличенных данных. Массив типа `byte` просто идеально подходит для этих целей.

Например, вам нужно куда-то скопировать файл.

Вам не нужно обрабатывать содержимое файла: вы просто хотите создать область памяти (буфер), скопировать в нее содержимое файла, а затем записать эти данные из буфера в другой файл. Массив типа `byte` — то, что нужно для этих целей.

Тем более, что в переменной-типа-массив хранится только ссылка на область памяти. При передаче значения этой переменной в какой-то метод произойдет только передача адреса в памяти, а сам блок памяти копироваться не будет.

```
byte[] buffer =newbyte[1024*1024];
FileInputStream sourceFile =new FileInputStream("c:\\data.txt");
FileOutputStream destFile =new FileOutputStream("c:\\output.tx
t");
while (true)
{
int size = sourceFile.read(buffer);   // читаем данные из файла в
буфер
   destFile.write(buffer, 0, size);      // записываем данные из
буфера в файл
```

```
   // прекращаем копирование, если буфер заполнен не полностью
if (size < buffer.length)break;
}
sourceFile.close();
destFile.close();
```

Вещественные типы

Среди примитивных типов также есть два вещественных. Хотя это не совсем точное название. Официально они называются ***числа с плавающей точкой — floating point numbers*.** Название происходит из стандарта, когда целую и дробную часть числа разделяет точка (а не запятая).

Тип `float`

Само название типа `float` происходит от ***floating point number***. Размер этого типа совсем небольшой — **всего 4 байта (32 бита**), но он может хранить значения от `-3.4*1038` до `3.4*1038`. Под мантиссу отдается 24 бита, под экспоненту — 8 бит. Этот тип способен хранить всего 8 значащих цифр.

Такой подход позволяет хранить гораздо большие числа, чем `int`, используя все те же 4 байта. Но при этом мы жертвуем точностью. Часть памяти расходуется на хранение мантиссы, поэтому такие числа хранят всего 6-7 знаков после запятой, остальные отбрасываются.

Пример:

Код	Значение переменной
`float a = ( float ) 123.456789;`	`123.45679`
`float a = ( float ) 12345.9999;`	`12346.0`
`float a = ( float ) -123.456789E-2;`	`-1.2345679`

Как видите, основной недостаток этого типа — **очень маленькое количество значащих цифр и потеря точности уже в восьмой цифре**. Поэтому тип `float` не сильно популярен среди Java-программистов.

Тип `double`

Тип `double` является стандартным типом с плавающей точкой. Его название происходит от ***double floating point*.** Его еще называют числом с плавающей точкой *двойной точности*. Все вещественные литералы по умолчанию имеют тип `double`.

Этот тип занимает **8 байт памяти (64 бита)** и может хранить значения от `-1.7*10308` до `1.7*10308`. Важным моментом является то, что под его мантиссу отводится 53 бита, а остальные 11 – под экспоненту.

Это позволяет хранить 15-17 значащих цифр.

Пример:

Код	Значение переменной
`double a = 1234567890.1234567890;`	`1234567890.1234567`
`double a = 1234567890.1234512345;`	`1234567890.1234512`
`double a = 1234567890.1357913579;`	`1234567890.1357913`

Такая точность, особенно в сравнении с типом `float`, является определяющей: 99% всех операций с вещественными числами выполняются с типом `double`.

Под экспоненту выделяется `11` бит, что позволяет хранить степень десятки от `-323` до `+308` (степень двойки — от `-1024` до `+1023`). Тип `double` легко может хранить число с сотней нулей после запятой:

Код	Значение переменной
`double a = 2E-300 * 3E+302`	`600.0`

Бесконечность

Числа с плавающей точкой обладают еще одной интересной особенностью: они позволяют хранить специальное значение, обозначающее **бесконечность**. Причем может быть **положительная бесконечность и отрицательная бесконечность**.

Примеры:

Код	Примечание
`System.out.println( 100.0 / 0.0 );`	`Infinity`
`System.out.println( -100.0 / 0.0 );`	`-Infinity`

`double a = 1d / 0d;`
`double b = a * 10;`
`double c = b - 100;`
	`a == Infinity`
	`b == Infinity`
	`c == Infinity`

Если бесконечность умножить на число, получится бесконечность. Если к бесконечности добавить число, получится бесконечность. Очень удобно.

Не число (`NaN`)

Числа с плавающей точкой могут хранить еще одно специальное значение — `NaN`. Это сокращение от ***Not a Number* (не число).**

В математике, если разделить бесконечность на бесконечность, должна возникнуть неопределенность.

Ну, а в Java, если разделить бесконечность на бесконечность, будет `NaN`.

Примеры:

Код	Примечание
`System.out.println(0.0 / 0.0);`	`NaN`
`double infinity = 1d / 0d;` `System.out.println(infinity / infinity);`	`NaN`

`double a = 0.0 / 0.0;`
`double b = a * 10;`
`double c = b - 100;`
`double d = a + infinity;`
	`a == NaN`
	`b == NaN`
	`c == NaN`
	`d == NaN`

Любая операция с `NaN` дает `NaN`.

Тип `char`

Среди примитивных типов в **Java** есть еще один, который заслуживает особого внимания — тип `char`. Его название происходит от слова *Character*, а сам тип используется для того, чтобы хранить символы.

Но еще интереснее тот факт, что тип `char` — это и числовой тип тоже! Так сказать, тип двойного назначения.

Все дело в том, что на самом деле тип `char` хранит не символы, а *коды символов* из кодировки Unicode. Каждому символу соответствует число — числовой код символа.

Каждая переменная типа `char` занимает в памяти два байта (как и тип `short`). Но в отличие от типа `short`, целочисленный тип `char` — беззнаковый, и может хранить значения от `0` до `65,535`.

Тип `char` — гибридный тип. Его значения можно интерпретировать и как числа (их можно складывать и умножать), и как символы. Так было сделано потому, что хоть символы и имеют визуальное представление, для компьютера они в первую очередь просто числа. И работать с ними как с числами гораздо удобнее.

> *Unicode* — это специальная таблица (кодировка), которая содержит все символы мира. И у каждого символа есть свой номер. Выглядит она примерно так:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00A0		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
00B0	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
00C0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00D0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
00E0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï

Присвоить значение переменной типа `char` можно разными способами.

Код	Описание
`char a = 'A';`	Переменная `a` будет содержать латинскую букву `A`.
`char a = 65;`	Переменная `a` будет содержать латинскую букву `A`. Ее код как раз `65`.
`char a = 0x41;`	Переменная `a` будет содержать латинскую букву `A`.Ее код как раз `65`, что равно `41` в шестнадцатеричной системе.
`char a = 0x0041;`	Переменная `a` будет содержать латинскую букву `A`.Ее код как раз `65`, что равно `41` в шестнадцатеричной системе. Два лишних нуля ничего не меняют.
`char a = '\u0041';`	Переменная `a` будет содержать латинскую букву `A`.Еще

	один способ задать символ по его коду.

Чаще всего просто указывают символ в кавычках (как в первой строке таблицы). Хотя популярен и последний способ. Его преимущество в том, что его можно использовать в строках.

И как мы говорили, тип `char` — это и целочисленный тип тоже, поэтому можно написать так:

Код	Вывод на экран

`char a = 'A';`
`a++;`
`System.out.println(a);`
	На экран будет выведена латинская буква `B`.Потому что: `A` – `65` `B` – `66` `C` – `67`

Тип `boolean`

И последний примитивный тип — это `boolean`.

Как вы уже знаете, он может принимать только два значения: `true` и `false`.