

1. Специализация: данные и функции

Экран	Слова
Титул	Перейдём к интересному: что можно хранить в джаве, как оно там хранится, и как этим манипулировать
На прошлом уроке	На прошлом уроке мы коротко поговорили об истории и причинах возникновения языка джава, вскользь посмотрели на инструментарий, который позволит нам писать на джава и получать результат, поверхностно изучили интерфейс командной строки, научились стремительно создавать довольно симпатичную документацию к своему коду и посмотрели на то как можно автоматизировать ручную работу при компиляции своих проектов.
На этой лекции	Будет рассмотрен базовый функционал языка, то есть основная встроенная функциональность, такая как математические операторы, условия, циклы, бинарные операторы. Далее способы хранения и представления данных в Java, и в конце способы манипуляции данными, то есть функции (в терминах языка называющиеся методами)
Слайд	Хранение данных в Java осуществляется привычным для программиста образом: в переменных и константах, желательно именованных, но об этом позже, для начала поговорим о том, какие вообще бывают языки относительно типов и собственно типы. Итак, языки программирования бывают типизированными и нетипизированными (бестиповыми). Про нетипизированные языки мы много говорить не будем, они не представляют интереса не только для джава программистов, но и в целом, в современном программировании.
Перфокарта	Отсутствие типизации в основном присуще чрезвычайно старым и низкоуровневым языкам программирования, например, Forth и некоторым ассемблерам. Все данные в таких языках считаются цепочками бит произвольной длины и, как следует из названия, не делятся на типы. Работа с ними часто труднее, и при чтении кода не всегда ясно, о каком типе переменной идет речь. При этом часто бестиповые языки работают быстрее типизированных, но описывать с их помощью большие проекты со сложными взаимосвязями довольно утомительно

Экран	Слова
<p>Java является языком со стро-гой (также можно встретить термин «сильной») явной статической типизацией</p>	<p>Что это значит?</p> <p>Статическая типизация означает, что у каждой переменной должен быть тип и мы этот тип поменять не можем. Этому свойству противопоставляется динамическая типизация, где мы можем назначить переменной сначала один тип, потом заменить на другой;</p> <p>Термин явная типизация говорит нам о том, что при создании переменной мы должны ей обязательно присвоить какой-то тип, явно написав это в коде. Бывают языки с неявной типизацией, например, Python, там можно как указать тип, так его и не указывать, язык сам попробует по контексту догадаться, что вы имели в виду;</p> <p>Строгая (или иначе сильная) типизация означает, что невозможно смешивать разнотипные данные. Тут есть некоторая оговорка, о которой мы поговорим позже, но с формальной точки зрения язык джава - это язык со строгой типизацией. С другой стороны, существует JavaScript, в котором запись <code>2 + true</code> выдаст результат 3.</p>
<p>таблица «Ос-новные типы данных»</p>	<p>Все данные в Java делятся на две основные категории: примитивные и ссылочные. Чтобы отправить на хранение какие-то данные используется оператор присваивания, который вам всем хорошо знаком.</p> <p>Думаю, не лишним будет напомнить, что присваивание в программировании - это не тоже самое, что математическое равенство, демонстрирующее тождественность, а полноценная операция. Все присваивания всегда происходят справа налево, то есть сначала вычисляется правая часть, а потом результат вычислений присваивается левой. Исключений нет, именно поэтому в левой части не может быть никаких вычислений.</p>


Экран	Слова
<p>Антипаттерн «магические числа»</p>	<p>Почти во всех сегодняшних примерах, которые мы будем писать, мы будем использовать антипаттерн - плохой стиль для написания кода. Числа, которые находятся справа от оператора присваивания будут использоваться в коде без пояснений. Такой антипаттерн называется "магическое число". Непонятно, что это за число, почему это число именно такое и что будет, если это число изменить. Так лучше не делать. О константности мы ещё поговорим, но заранее нужно сказать, что рекомендуется помещать числа в константы, которые хранятся в начале файла.</p> <p>Плюсом такого подхода является возможность легко корректировать значения переменных в достаточно больших проектах. Представьте, что в вашем коде несколько тысяч строк, а какое-то число, скажем, возраст совершеннолетия, число 18, использовалось несколько десятков раз. При использовании приложения в стране, где совершеннолетием считается 21 год вы должны будете перечитывать весь код в поисках магических "18" и править их на "21". Да ещё и не запутаться, те ли это 18, которые про совершеннолетие, а не про количество карманов в жилетке Вассермана, хоть мы и знаем, что их 26. В случае с константой изменить число нужно в одном месте.</p>
<p>таблица «Основные типы данных»</p>	<p>Примитивных типов всего восемь и это, наверное, первое, что спрашивают на джуниорском собеседовании, это байт, шорт, инт, лонг, флоут, дабл, чар и булин. как вы можете заметить в этой таблице, шесть из восьми типов имеет диапазон значений, а значит основное их отличие в объёме занимаемой памяти. На самом деле у дабла и флоута тоже есть диапазоны, просто они заключаются в другом и их довольно сложно отобразить в простой таблице. Что значат эти диапазоны? они значат, что если мы попытаемся положить в переменную меньшего типа какое-то большее значение, произойдёт неприятность, которая носит название «переполнение переменной».</p>
<p>переполнение переменной если презы умеют в гифки, нужна вода, льющаяся в переполненный стакан</p>	<p>Интересное явление, рассмотрев его мы рассмотрим одни из самых трудноуловимых ошибок в программах, написанных на строго типизированных языках. С переполнением переменных есть одна неприятность: их не распознаёт компилятор. Итак, переполнение переменной - это ситуация, в которой как и было только что сказано, мы пытаемся положить большее значение в переменную меньшего типа. чем именно чревато переполнение переменной легче показать на примере (тут забавно будет вставить в слайд пару-тройку картинок из вот этого описания расследования крушения ракеты из-за переполнения переменной https://habr.com/ru/company/pvs-studio/blog/306748/)</p>


Экран	Слова
Лайвкод	<p>если мы создадим переменную скажем байт, диапазон которого от -128 до +127, и присвоим этой переменной значение, скажем, 200, что произойдёт? правильно, переполнение, как если попытаться влить пакет молока в напёрсток, но какое там в нашей переменной останется значение максимальное 127? 200-127? какой-то мусор? именно этими вопросами никогда не надо задаваться, потому что каждый язык, а зачастую и разные компиляторы одного языка ведут себя в этом вопросе по разному. лучше просто не допускать таких ситуаций и проверять все значения на возможность присвоить их своим переменным. В современном мире гигагерцев и терабайтов почти никто не пользуется маленькими типами, их, наверное, можно считать своего рода пережитком, но именно из-за этого ошибки переполнения переменных становятся опаснее испанской инквизиции, их никто не ожидает (тут не помешает кадр из Monty Python «no one expects spanish inquisition»)</p>
Слайд	Задание для самопроверки:
Бинарное (битовое) представление	<p>При разговоре о переполнении нельзя не упомянуть о том, что же именно переполняется. Несколько минут поговорим о единичках и ноликах в целом, без контекста джавы. Важно помнить, что все компьютеры так или иначе работают от электричества и являются довольно примитивными по сути устройствами, которые понимают только два состояния: есть напряжение в электрической цепи или нет. Эти два состояния принято записывать в виде 1 и 0, соответственно. Отсюда и пошла вся весьма увлекательная работа с бинарными данными. С помощью одних лишь единиц и нулей научились вон какие штуки делать, компьютерами назвали. Итак, все данные в любой программе до изобретения квантовых компьютеров - это единицы и нули. Данные в программе на джава не исключение, и удобнее всего это явление рассматривать, естественно, на примере примитивных данных, о которых мы сейчас и говорим.</p>
Слайд	<p>Далее будут представлены сведения которые касаются не только языка Java но и любого другого языка программирования эти сведения помогут нам разобраться в деталях того как хранится значение переменной в программе и как в целом происходит работа компьютерной техники. А поскольку мы можем оперировать только двумя значениями то мы вынуждены использовать то что называется двоичной системой счисления.</p>

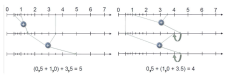
Экран	Слова
Слайд с числами в 10 и 2	Двоичная система счисления это такая же система счисления как привычные нам десятичная, но с основанием два то есть в привычной нам десятичной системе мы знаем 10 цифр 0123456789 всё остальное это так или иначе составленные из этих цифр числа а в двоичной системе счисления таких цифр только две ноль и один и всё остальное будут составленные из этих цифр числа.
Слайд с числами в 10 2 и 16	Существуют и другие системы счисления раньше была очень популярна восьмеричную систему сейчас она отходит на второй план полностью уступая свое место шестнадцатичной системе счисления которая кстати тоже часто используется в компьютерной технике но сейчас не об этом.
Слайд	И каждая цифра в десятичной записи числа называется разрядом собственно в двоичной записи чисел каждая цифра тоже называется разрядом но для компьютерной техники и этот разряд называется битом то есть это 1 б информации либо ноль либо единицу эти биты принято собирать в группы по восемь штук по восемь разрядов эти группы по восемь разрядов называются байт то-есть в языке Java мы можем оперировать Минимальный единицы информации такой как байт для этого даже есть соответствующий тип который так и называется.
Слайд	Внимательный зритель мог обратить внимание что я обозначил диапазон байта как числа от -128 до +127 и не сложно посчитать что 8 байт информации могут в себе содержать ровно 256 значений то есть как раз диапазон от -128 до +127 само число 127 в двоичной записи это семиразрядное число, все разряды которого единицы. Последний восьмой самый старший бит определяет знак числа.
Слайд	Здесь можно начать долгий и скучный разговор о схематехнике и хранении отрицательных чисел с применением техники дополнительного кода, но нам достаточно будет знать формулу расчёта записи отрицательных значений. нам нужно в прямой записи поменять все нули на единицы и единицы на нули, и поставить старший бит в единицу, чтобы получить на единицу меньшее отрицательное число, так 0 будет -1, 1 будет -2, 2 станет -3 и так далее.
таблица «Основные типы данных»	Соответственно, числа больших разрядностей могут хранить большие значения, как мы видели на примере целочисленных типов, теперь нехитрое преобразование диапазонов из десятичной системы счисления в двоичную покажет что байт это один байт, шорт это два байта, то есть 16 бит, инт это 4 байта то есть 32 бита, а лонг это 8 байт или 64 бита хранения информации.

Экран	Слова
Слайд	<p>Внезапно, становится очевидно что такое переполнение и почему оно так называется: мы не можем записать в переменную больше битов чем она в состоянии хранить поэтому биты которые находится левее размерности переменной просто будут отброшены им негде будет храниться также как если бы мы переполнили стакан, вода просто переливается за граница стакана. Остается вопрос что осталось в стакане, когда переполнявшая его вода вылилась наружу? Явно не то, что мы ожидаем, также и с переменными, если положить в них переполняющее значение мы точно не увидим там что-то ожидаемое.</p>
Слайд	<p>Мы иногда будем возвращаться к бинарным представлениям и разным системам счисления, что называется для общего развития.</p>
Слайд	<p>Задание для самопроверки:</p>
<p>таблица «Основные типы данных»</p>	<p>целочисленных типов аж 4 и они занимают 1,2,4,8 байт соответственно. про char, несмотря на то, что он целочисленный мы поговорим чуть позднее. с четырьмя основными целочисленными типами всё просто - значения в них могут быть только целые, никак и никогда невозможно присвоить им дробных значений, хотя и тут можно сделать оговорку и поклон в сторону арифметики с фиксированной запятой, но мы этого делать не будем, чтобы не взрывать себе мозг и не сбиваться с основной мысли. итак, целочисленные типы с диапазонами минус 128 плюс 127, минус 32768 плюс 32767, я никогда не запомню что там после минус и плюс 2млрд, и четвёртый, который лично я никогда даже не давал себе труд дочитать до конца про эти типы важно помнить два факта: инт - это самый часто используемый тип, если сомневаетесь, какой использовать, используйте инт. все числа, которые вы пишете в коде - это инты, даже если вы пытаетесь их присвоить переменной другого типа</p> <p>Я вот сказал, что инт самый часто используемый и внезапно подумал: а ведь на практике, чаще всего, было бы достаточно шорта, например, в циклах, итерирующихся по подавляющему большинству коллекций, или при хранении значений, скажем, возраста человека, но всё равно все по привычке используют инт</p>

Экран	Слова
<pre>byte b = 120; byte b1 = 200;</pre>	<p>Теперь плавно подошли к тому что все написанное нами в коде программы цифры - это по умолчанию интеджеры, а дробные даблы, и становится достаточно интересно как они преобразуются например в меньше типы. Тут все просто - если мы пишем цифрами справа число, которое может поместиться в меньший тип слева то статический анализатор кода его пропустит, а компилятор преобразует в меньший тип автоматически. Как мы видим, к маленькому байту вполне успешно присваивается инт. получается, обманул, сказав, что все числа это инты?</p> <p>если же мы пишем число которое больше типа слева и соответственно поместиться не может, среда разработки нам выдает сообщение о том что произойдет переполнение и вообще невозможно положить так много в такой маленький контейнер. А в предупреждении компилятора мы видим, что ожидался байт, а даём мы инт.</p>
<p>лайвкод в котором нужно показать попытку присвоения лонга, показать предупреждения среды</p>	<p>Интересное начинается когда мы хотим записать в виде числа какое-то значение большее чем может принимать инт, и явно присвоить начальное значение переменной типа лонг. давайте посмотрим на следующий пример - попытку присвоить значение 5 млрд переменной типа лонг. помним, что в лонге можно хранить очень большие числа, мы то явно видим, что слева лонг и точно знаем что присваиваемое значение в него поместится, но среду и компилятор это почему-то мало волнует, значит и тут наврал? давайте разбираться по порядку: если мы посмотрим на ошибку, там английскими буквами будет очень понятно написано - не могу положить такое большое значение в переменную типа инт. а это может значить только одно: справа - инт. не соврал. Почему большой инт без проблем присваивается к маленькому байту? поговорим буквально через несколько минут, пока просто запомним, что это происходит</p>
<pre>long l0 = 3_000_000_000L; float f = 0.123f;</pre>	<p>Аналогичная ситуация с флоутами и даблами. Все дробные числа, написанные в коде - это даблы, поэтому положить их во флоут без дополнительных плясок с бубном невозможно по понятной причине, они туда не помещаются. В этих случаях к написанному справа числу нужно добавить явное указание на его тип, то есть мы как бы говорим компилятору, что мы точно уверены, что справа большой лонг (или маленький флоут, в зависимости от контекста). Это уводит нас в сторону разговора о преобразовании типов, опять же, поговорим об этом через несколько минут, чтобы не отвлекаться от хранения примитивов. Просто запоминаем, что для лонга пишем Л а для флоута ф. Чаще всего л пишут заглавную, чтобы подчеркнуть, что тип большой, а ф пишут маленькую, чтобы подчеркнуть, что мы уменьшаем тип. но регистр значения не имеет, можно писать и так и так.</p>

Экран	Слова
<p>Слайд</p>	<p>Далее речь пойдёт о том, что называется числами с плавающей запятой. в англоязычной литературе эти числа называются числа с плавающей точкой (от английского флоутин поинт), такое различие связано с тем, что в русскоязычной литературе принято отделять дробную часть числа запятой, а в европейской и американской - точкой.</p> <p>Как мы видим, два из восьми типов не имеют диапазонов значений, это связано с тем, что диапазоны значений флоута и дабла заключаются не в величине возможных хранимых чисел, а в точности этих чисел после запятой. до какого знака будет сохранена точность. Говорить о числах с плавающей точкой и ничего не сказать об особенностях их хранения - преступление, поэтому, снова немного отвлечёмся на общее развитие, несколько минут поговорим не о джаве, а о компьютерах в целом.</p>
<p>много хорошо и подробно, но на С https://habr.com/ru/post/112953/</p> <p>Форматы с плавающей запятой</p> 	<p>Работает по стандарту IEEE 754 (1985 года). Для работы с числами с плавающей запятой на аппаратурном уровне к обычному процессору который находится в вашем устройстве ещё прикручивают математический сопроцессор, он нужен, чтобы постоянно вычислять эти ужасные плавающие запятые. Если попытаться уложить весь стандарт в два предложения, то получится примерно следующее: формат подразумевает три поля (знак, 8(11) разрядов поля порядка, 23(52) бита мантисса). Чтобы получить из этой битовой каши число надо -1 возвести в степень знака, умножить на 2 в степени порядка минус 127 и умножить на $1 +$ мантиссу делённую на два в степени размера мантиссы. Формула на экране, она не очень сложная. В остальном, ничего не понятно, но очень интересно, понимаю, давайте попробуем на примере.</p>
<p>возьмём число $+0,5$</p>	<p>с ним всё довольно просто, чтобы получить $+0,5$ нужно 2 возвести в -1 степень. поэтому, если развернуть обратно формулу, описанную выше, в знак и мантиссу мы ничего не пишем, оставляем 0, а в порядке должно быть 126, тогда мы должны будем -1 возвести в 0ю степень и получить положительный знак, умножить на 2 в степени $126 - 127 = -1$, получив внезапно $0,5$ и умножить на 1 плюс пустая мантисса, в которой по сути не очень важно, что делить, что умножать и в какие степени возводить, всё равно 0 будет. Отсюда становится очевидно, что чем сложнее мантисса и чем меньше порядок, тем более точные и интересные числа мы можем получить.</p>

Экран	Слова
<p>а что если - 0,15625</p>	<p>Попробуем немного сложнее: число $-0,15625$, чтобы понять как его записывать, откинем знак, это будет единица в разряде, отвечающем за знак, и посчитаем мантиссу с порядком. представим число как положительное и будем от него последовательно отнимать числа, являющиеся отрицательными степенями двойки, чтобы получить максимально близкое к нулю значение</p>
$2^1 = 2 \quad 2^0 = 1.0$ $2^{-1} = 0.5$ $2^{-2} = 0.25 \quad 2^{-3} = 0.125$ $2^{-4} = 0.0625$ $2^{-5} = 0.03125$ $2^{-6} = 0.015625$ $2^{-7} = 0.0078125$ $2^{-8} = 0.00390625$	<p>получается, что -1 и -2 степени отнять не получится, мы явно уходим за границу нуля, а вот -3 прекрасно отнимается, значит порядок будет $127 - 3 = 124$, осталось понять, что получается в мантиссе. видим, что оставшееся после первого вычитания число - это 2 в -5 степени. значит в мантиссе мы пишем 01 и остальные нули, то есть слева направо указываем, какие степени после -3 будут нужны. -4 не нужна, а -5 нужна. Получится, что</p>
$(-1)^1 \times 2^{(124-127)} \times (1 + \frac{2097152}{2^{23}}) = 1,15652$ $(-1)^1 \times 1,01e-3 = 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} = 1 \times 0,125 + 0 \times 0,0625 + 1 \times 0,03125 = 0,125 + 0,03125 = 0,15625$	<p>так наше число можно посчитать двумя способами: по приведённой на слайде формуле или последовательно складывая разряды мантиссы умноженные на двойку в степени порядка, уменьшая порядок на каждом шагу, как это показано на слайде</p>
<p>Особенности Числа с плавающей запятой</p> 	<p>Ну, что, поковырялись в детальках и винтиках, можно коротко поговорить об особенностях чисел с плавающей точкой, а именно: в числах с плавающей точкой бывает как положительный, так и отрицательный ноль, в отличие от целых чисел, где ноль всегда положительный; у чисел с плавающей запятой есть огромная зона, отмеченная на слайде, которая являет собой непредставимые числа слишком большие для хранения внутри такой переменной или настолько маленькие, что мнимая единица в мантиссе отсутствует; в таком числе можно хранить значения положительной и отрицательной бесконечности; при работе с такими числами появляется понятие не-числа, при этом важно помнить, что $\text{NaN} \neq \text{NaN}$, а если очень сильно постараться, можно хранить там собственные данные, но это выходит далеко за пределы курса, и используется в каких-нибудь чрезвычайно маломощных процессорах для цифровой обработки сигналов, например</p>

Экран	Слова
<p>Проблемы Пример вычислений</p> 	<p>у чисел с плавающей запятой могут иногда встречаться и проблемы в вычислениях, пример на слайде чрезвычайно грубый, но при работе, например, со статьями или миллионными долями во флоте, с такой проблемой вполне можно столкнуться. порядок выполнения действий может влиять на результат выполнения этих действий, что противоречит математике</p>
Слайд	Задание для самопроверки:
<p>таблица «Основ- ные типы данных»</p>	<p>Казалось бы, это было так давно, но вернёмся к нашей таблице с примитивными типами данных. Что ещё важного мы видим в этой таблице? шесть из восьми примитивных типов могут иметь как положительные, так и отрицательные значения они называются одним словом «знаковые» типы. Можно заметить что в таблице есть два типа, у которых есть диапазон но нет отрицательных значений, это булин и чар. По порядку с простого, булев тип хранит true и false, тут я вам ничего нового не скажу, на собеседах иногда спрашивают сколько места он занимает, в Java объём хранения не определён и зависит от конкретной JVM, обычно считают, что это байт, несмотря на то что хватает и бита, но тогда значительно усложнятся алгоритмы хранения и доступа в коллекциях, но беседа об этом ведёт нас в сложные дебри оптимизации адресации памяти и прочих регистровых алгоритмов и структур данных</p>
таблица UTF-8	<p>Что касается чара я нахожу его самым интересным примитивным типом данных. Он единственный беззнаковый целочисленный тип в языке, то есть его старший разряд хранит полезное значение, а не признак положительности. Тип целочисленный но по умолчанию среда исполнения интерпретирует его как символ по таблице utf-8. Таблицу несложно найти в интернете и хранимое в чаре число является индексом в этой таблице, а значит совершенно точно не может быть отрицательным</p>
Слайд	<p>Вы конечно же об этом уже знаете но именно сейчас важно дополнительно упомянуть о том что в языке Java есть разница между одинарными и двойными кавычками. В одинарных кавычках мы всегда записываем символ который на самом деле является целочисленным значением а в двойных кавычках мы всегда записываем строку, которая фактически является экземпляром класса String. Поскольку типизация строгая то мы не можем записывать в чары строки а в строки числа</p>

Экран	Слова
Слайд	<p>С типизации вроде разобрались давайте разберёмся с основными понятиями чтобы больше в них никогда не путаться в Java как и в любом другом языке программирования есть три основных понятия, связанных с данными переменными и использованием значений. это объявление присваивание инициализация. Для того чтобы объявить переменную нужно написать её тип и название также часто вместо названия можно встретить термин идентификатор. Далее в любой момент можно присвоить этой переменной значение то есть необходимо написать идентификатор использовать оператор присваивания, который выглядит как обычный знак равенства и справа написать значение которое вы хотите присвоить данной переменной, поставить в конце строки точку с запятой. статический анализатор кода автоматически проверит соответствие типов и при выполнении программы значение будет присвоено. Также существует понятие инициализации - это когда объединяются на одной строке объявление и присваивание. Всё довольно просто и прозрачно.</p>
преобразование типов	<p>джава это язык со строгой статической типизацией, но преобразование типов в ней всё равно есть. прямо сейчас нам имеет смысл поговорить о преобразовании примитивных типов. преобразование типов - это когда компилятор видит, что типы переменных по разные стороны присваивания разные, помнит, что типизация статическая, и надо как-то решать это противоречие. В разговоре или в сообществах можно услышать термины тайпкастинг, кастинг, каст, кастануть, и другие производные от англ тайпкастинг.</p>
Слайд	<p>Преобразование типов бывает явное и неявное. Начнём с более простого, неявного. Неявное преобразование типов мы с вами уже даже могли заметить, когда присваивали числа всяким маленьким переменным. Припоминаем, что число справа это инт, а значит 32 разряда, а слева было что-то небольшое, например, байт, и в нём всего 8 разрядов, но ни среда ни компилятор на нас не поругались. потому что значение в большом инте не превысило 8 разрядов маленького байта, и компилятор решил, что раз уж у него все числа в тексте программы инты - нет смысла ругаться на пользователя, он всё равно ничего не исправит.</p>
Слайд	<p>Итак неявное преобразование типов происходит в случаях, когда, если можно так выразиться, всё и так понятно. В случае, если неявное преобразование невозможно, статический анализатор кода выдаёт нам ошибку, что ожидался один тип, а мы даём другой.</p>

Экран	Слова
Слайд	Явное преобразование типов - это когда мы прямо пишем в коде, что такая-то переменная должна иметь такой-то тип. Этот вариант типизации мы уже тоже видели, когда дописывали к лонгу L а к флоту f. Но чаще всего случается, что мы присваиваем переменным не те значения, которые пишем в тексте программы, а те, которые получились в результате каких-то вычислений. И часто эти результаты вычислений хранятся в более больших переменных.
<code>int i = 10; byte b = i;</code>	вот простейший пример, мы совершенно точно уверены, что внутри переменной <code>i</code> лежит значение не превышающее байт по хранению, и допустим, что точно знаем, что по нашим бизнес-задачам ни в каких вычислениях никогда не сможет это значение превысить, а значит мы можем явно сообщить компилятору, что значение точно поместится в байт. явно преобразовать типы.
<code>byte b = (byte)i ;</code>	для явного преобразования типов нужно в правой части оператора присваивания перед идентификатором переменной в скобках добавить название типа, к которому мы хотим преобразовать значение этой переменной Преобразование типов, очевидно, затрагивает не только примитивные типы, но и ссылочные, но не будем забегать вперед, пока остановимся на этом.
константность	Последнее о примитивах на сегодня - это константность. поскольку мало кто сегодня хорошо умеет в латынь скажу, констаре - это глагол, означающий стоять твёрдо. а значит для математики и языков программирования константность это свойство неизменяемости. в джаве ключевое слово <code>const</code> не реализовано, хоть и входит в список ключевых. константы же можно создавать при помощи ключевого слова <code>final</code> , что решает довольно забавный парадокс терминологии, ведь от программистов на других языках часто можно часто услышать словосочетание константная переменная, а здесь прямой перевод говорит, что это переменная с конечным значением.
Слайд	Конечно, ключевое слово <code>final</code> возможно применять не только с примитивами, но и со ссылочными типами, методами, классами, но как и в случае с преобразованием типов - не будем забегать вперед. Пока просто запомним - переменная или идентификатор с конечным значением, именно такая формулировка нам поможет с константностью объектов.
Слайд	Задание для самопроверки:
Слайд	Разобравшись с примитивными типами данных мы можем переходить к ссылочным помните я в самом начале говорил что есть два больших вида данных примитивные и Ссылочное вот примитивных восемь а ссылочные это все остальные и это скорее хорошая новость чем плохая потому что не надо запоминать их бесконечные названия.

Экран	Слова
Слайд	Самым простым из ссылочных типов является массив. фактически массив выведен на уровень языка и не имеет специального ключевого слова как названия хотя если копнуть гораздо глубже то можно увидеть что у него есть внутреннее название слово эррэй с большой буквы обрамлённое двумя символами нижнего подчёркивания с каждой стороны. Не буду утомлять вас скучной частью о назначении массива и тем что там хранятся наборы однотипных данных, сразу к делу.
Слайд	Ссылочные типы отличаются от примитивных местом хранения информации. в примитивах данные хранятся прямо там, где существует переменная и где написан её идентификатор, а по идентификатору ссылочного типа внезапно хранится не значение, а ссылка. Эту ссылку можно представить как ярлык у вас на рабочем столе, то есть очевидно, что непосредственная информация хранится не там, где написан идентификатор. Такое явное разделение идентификатора переменной и данных, которые по нему можно найти, нам будет важно помнить и понимать при работе с ООП на дальнейших занятиях.
Слайд	в идентификаторе массива, фактически, находится адрес его первого элемента. не лишним будет напомнить, что массив - это единая сплошная область данных, в связи с чем в массивах возможно осуществление доступа по индексу. Самый младший индекс любого массива - ноль, поскольку индекс - это значение смещения по элементам относительно начального адреса массива. То есть, для получения самого первого элемента нужно сместиться на ноль шагов. Очевидно, что самый последний элемент в массиве из, скажем, десяти значений, будет храниться по девятому индексу.
Лайвкод	Массивы в джава создают несколькими способами. В общем виде объявление это тип квадратные скобки как обозначение того, что это будет массив из переменных этого типа, идентификатор. Инициализировать массив можно либо ссылкой на другой массив, пустым массивом или заранее заданными значениями, записанными через запятую в фигурных скобках. Присвоить в процессе работы идентификатору возможно только значение ссылки из другого идентификатора или новый пустой массив.
Слайд	Давайте, раз уж заговорили о том как создавать поговорим об ограничениях, накладываемых на это действие. Никак и никогда нельзя присвоить идентификатору целый готовый массив в процессе работы, нельзя стандартными средствами переприсвоить ряд значений части массива (так называемые слайсы или срезы). Кстати, получить стандартными средствами срез массива на чтение тоже, к сожалению, нельзя.

Экран	Слова
Слайд	Массивы бывают как одномерные, так и многомерные и тут важно помнить, что многомерный массив - это всегда массив из массивов меньшего размера: двумерный массив - это массив одномерный, трёхмерный - массив двумерных и так далее. Правила инициализации у них не отличаются.
Слайд	преобразовать тип массива нельзя никогда, каждого отдельного элемента можно, а массива целиком нельзя, это связано с тем, что под массивы сразу выделяется непрерывная область памяти, а со сменой типа всех значений массива эту область нужно будет или значительно расширять или значительно сужать, а такую ответственность JVM никогда на себя не возьмёт. Но можно преобразовать тип каждого отдельного элемента массива после его чтения так, как мы это уже делали с примитивами.
Слайд	если мы пишем ключевое слово <code>final</code> с массивом то это совершенно не значит, что мы не сможем изменить значения по индексам этого массива. ключевое слово <code>final</code> работает только с идентификаторами.
Слайд	Есть интересная особенность, она же отличительная черта языка от например C++, она заключается в том, что если вы планируете создавать нижние измерения массива в процессе работы программы, то при инициализации массива верхнего уровня вам не следует указывать размерности нижних уровней. Это связано с тем, что при инициализации джава сразу выделяет память под все измерения, а присваивание нижним измерениям новых значений всё равно будет пересоздавать области памяти, получается небольшая утечка памяти, не критично, но неприятно. Это, кстати, позволяет создавать не прямоугольные массивы.
Слайд	Прочитать из массива значение возможно обратившись к ячейке массива по индексу. Записать в массив значение возможно обратившись к ячейке массива по индексу, и применив оператор присваивания, тут ничего нового. В каждом объекте массива есть специальное поле, которое обозначает длину данного массива. Поле находится в классе <code>Array</code> и является публичной константой. Это, кстати, хорошо объясняет, почему нужно сразу объявлять длину массива и нельзя менять его значение, вдруг изменится длина, а поле с длиной константно.
Слайд	Задание для самопроверки:
Слайд	О данных пока хватит, предлагаю поговорить о базовой функциональности языка, то есть об арифметике, условиях, циклах и бинарных операторах. Здесь будем коротко, потому как совсем уж узких мест тут не предвидится

Экран	Слова
Слайд	<p>Математические операторы работают как и предполагается - складывают, вычитают, делят, умножают, делают это по приоритетам известным нам с пятого класса, а если приоритет одинаков - слева направо. Специального оператора возведения в степень как в пайтоне нет. Единственное, что следует помнить, что оператор присваивания продолжает быть оператором присваивания, а не является математическим равенством, а значит сначала посчитается всё, что слева, а потом результат попытается присвоиться переменной справа. Припоминаем что там за дела с целочисленным делением и отбрасыванием дробной части.</p>
лайвкод	<p>с условиями тоже всё достаточно прозрачно, никаких подводных камней, «если», «иначе если», «в противном случае». раз уж мы тут в шестерёнках копаемся, наверное, следует упомянуть, что конструкция от иф до элс является единым оператором выбора, какой бы длинный он ни был, поэтому надеяться что какие-то входные данные позволят выполниться двум веткам условного оператора немного самонадеяно. каждая ветвь условного оператора - это отдельный кодовый блок со своим окружением и локальными переменными.</p>
лайвкод	<p>Неплохой альтернативой многоступенчатого оператора иф-элс является оператор свич, который позволяет осуществлять множественный выбор между числовыми значениями. Оператор хорош почти всем, но у него есть ряд особенностей. первая и самая очевидная - это оператор, состоящий из одного кодового блока, то есть сегменты кода находятся в одной области видимости</p>
лайвкод	<p>если не поставит брейки - будем проваливаться в следующие кейсы, хотя иногда это нужно, но чаще всего является неожиданным поведением. нельзя создать диапазон значений, в отличие от иф-чика. нельзя создать локальные переменные с одинаковым названием для каждого кейса.</p>
Слайд	<p>Циклы представлены в джаве тремя братьями вайл, дувайл и фо. у них также нет каких-то сверхспецифичных особенностей, цикл он и есть цикл, набор повторяющихся действий до наступления условия. вайл - самый простой и примитивный, чаще всего используется, когда нужно описать простой бесконечный цикл. дувайл единственный цикл с постусловием, то есть сначала сделали, а потом решили нужно ли делать ещё раз. используется для ожидания ответов на заданный вопрос и возможного перезапроса по условию. фо - классический счётный цикл, его почему-то программисты любят больше всего.</p>

Экран	Слова
Слайд	есть ещё один, активно пропагандируемый цикл - это форич, но я его специально не отнёс к классическим собратьям, поскольку он работает немного неочевидным образом, о чём мы обязательно поговорим позже, поскольку для понимания его работы нам нужно как минимум ознакомиться с ООП и понятием Итератора.
Слайд	Бинарные операторы. В современных реалиях мегамоощных компьютеров вряд ли кто-то задумывается об оптимизации скорости выполнения программы или экономии занимаемой памяти. Но всё меняется, когда программист впервые принимает сложное решение: запрограммировать микроконтроллер или другой «интернет вещей». Там в вашем распоряжении жалкие пара сотен килобайт памяти, если очень повезёт, в которые нужно не только как-то впихнуть текст программы и исполняемый бинарный код, но и какие-то промежуточные, пользовательские и другие данные, буферы обмена и обработки. Другая ситуация, в которой нужно начинать «думать о занимаемом пространстве» это разработка протоколов передачи данных, чтобы протокол был быстрый, не гонял по сети сумасшедшие мегабайты данных и быстро преобразовывался. На помощь приходит натуральная (если можно так выразиться) для информатики система счисления, двоичная, мы о ней говорили в самом начале этой лекции.
Слайд	Манипуляции двоичными данными представлены в Джава следующими операторами: битовые и-или-не-ксор, сдвиг влево-вправо. литеральные и-или-не вам уже знакомы по условным операторам. Литеральные операции применяются ко всему числовому литералу целиком, а не к каждому отдельному биту. У них особенность заключается в том, как язык программирования интерпретирует числа. В Java, например, в таких операциях может участвовать только тип boolean, а C++ воспринимает любой ненулевой числовой литерал как истину, а нулевой, соответственно, как ложь. Логика формирования значения при этом остаётся такой же, как и при битовых операциях.
Таблицы истинности	Когда говорят о битовых операциях волей-неволей появляется необходимость поговорить о таблицах истинности. На слайде вы видите таблицы истинности для арифметических битовых операций.
Примеры в столбик	Битовые операции отличаются тем, что для неподготовленного взгляда они производят почти магические действия. а всё потому, что манипулируют двоичным представлением числа. На слайде вы видите примеры работы арифметических двоичных операторов, если очень внимательно посмотреть. можно увидеть, что соблюдаются показанные только что таблицы истинности

Экран	Слова
0-1-2-4-8-16-32-64-...	С битовыми сдвигами всё куда интереснее, они производят арифметический сдвиг значения слева на количество разрядов, указанное справа, на слайде вы видите числа, интересное свойство этих чисел состоит в том, что в битовом представлении это одна единственная единица, находящаяся в разных разрядах числа.
Слайд	Заодно это демонстрация сдвига на один разряд влево, и, как следствие, умножение на два. Таким образом, можем сделать вывод о том, что для умножения на 2 в степени N нужно число сдвинуть на N разрядов влево
...-64-32-16-8-4-2-1-0	Обратная ситуация со сдвигом вправо, он является целочисленным делением на 2 в степени N. Весьма удобно и быстро, если не нужны дробные части. Кстати, если посмотреть любой ассемблер любого коммерческого процессора, то можно увидеть, что все умножения и целочисленные деления в коде преобразованы в серию сдвигов и сложений, поскольку выполняются такие операции на простейших АЛУ процессора, без задействования математического сопроцессора, а значит, в разы, а иногда и в десятки раз быстрее.
$X \&\& Y = X Y =$ $!X = N \ll$ $K = N * 2KN \gg$ $K = N / 2Kx \& y =$ $.1x == 1y ==$ $1x y = .1x ==$ $1y == 1 x =$ $.1x == 0x = .1xy$	Резюмируем, ... читаем слайд
Слайд	Задание для самопроверки: — какое значение будет содержаться в переменной a после выполнения строки <code>инт a = 10.0ф/3.0ф</code>
Слайд	Функция - это исполняемый блок кода на языке джава. но в джаве функций не бывает. потому что всё, что мы пишем - находится в классах, даже первый хелловорлд, а функция, принадлежащая классу называется методом. Привыкаем к терминологии, в джаве - только методы, вне классов существовать то есть, что называется, висеть в воздухе, такие блоки кода не могут.

Экран	Слова
Слайд	Про функции (и их частный случай в Джаве - методы) важно помнить, что у них - параметры. не аргументы. аргументы - это у вызова функции. а вот у функций - параметры. У функций есть правила именования: функция - это переходный глагол совершенного вида в настоящем времени (вернуть, посчитать, установить, создать), часто снабжаемый дополнением, субъектом действия. методы в джаве пишутся lowerCamelCase то есть первая буква строчная а далее каждое новое слово с большой буквы.
Слайд	Методы обособлены и их параметры локальны, то есть никакой параметр никакой функции не виден другой функции. Из этого свойства есть очевидное следствие - нельзя писать функции внутри функций. Одна из наиболее частых ошибок новичка - невнимательно посмотреть на открывающие/закрывающие скобки и сильно удивляться тому, что компилятор ругается на синтаксически верно написанную функцию
Слайд	Все аргументы в джава передаются копированием, не важно, копирование это числовой константы, числового значения переменной или хранимой в переменной ссылке например на массив. Обратите внимание, что сам объект в метод не копируется, а копируется только его ссылка. Через несколько лекций мы поговорим о том, что в метод можно передать и другой метод, но пока что оставим эту магию в стороне.
Вот тут может мне какой-то реквизит понадобится вроде одноразовых тарелок, буду стек вызовов показывать	Возвращаемые из методов значения возникают ровно в том месте, где метод был вызван, это обусловлено архитектурой компьютеров общего назначения и так называемым стеком вызовов. Если мы вызываем несколько методов, а именно это мы чаще всего и делаем в наших программах, то весь контекст исполнения первого метода сохраняется, кладётся (на стек) в стопку уже вызванных методов и процессор идёт выполнять только что вызванный второй метод. по завершении вызванного второго метода мы снимаем со стека лежащий там контекст первого метода, кладём в него вернувшееся из второго метода значение, если оно есть, и продолжаем исполнять первый метод.
Слайд	
Слайд	
Слайд	
Цель учения — достичь наибольшего удовлетворения в получении знаний. Сюнь-цзы	Надеюсь, на этой лекции вам не было скучно и это не было только лишь повторением того что вы уже знаете. Я постарался добавить максимум полезной информации о всяких шестерёнках под капотом языка, поэтому, закончить хотелось бы словами классика, которые вы видите на слайде.