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 изучилит 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 безтиповые языки работают быстрее типизированных, но описывать с их помощью большие проекты со сложными взаимосвязями довольно утомительно

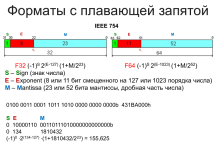
Экран	Слова
Java является языком со **строгой** (также можно встретить термин «**сильной**») **явной статической** типизацией	Что это значит? Статическая типизация означает, что у каждой переменной должен быть тип и мы этот тип поменять не можем. Этому свойству противопоставляется динамическая типизация, где мы можем назначить переменной сначала один тип, потом заменить на другой; 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 попробуед по контексту догадаться, что вы имели ввиду; 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 2 + true выдаст результат 3.
таблица «Основные типы данных»	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. 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. 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».

Экран	Слова
переполнение переменной если презы умеют в гифки, нужна вода, льющаяся в переполненный стакан	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/)
Лайвкод	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 монти пайтон «no one expects spanish inqisition»)
Бинарное (битовое) представление	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.

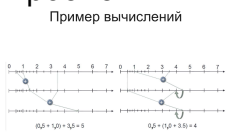
Экран	Слова
Слайд	Далее будут представлены сведения которые касаются не только языка Java но и любого другого языка программирования эти сведения помогут нам разобраться в деталях того как хранится значение переменной в программе и как в целом происходит работа компьютерной техники. А поскольку мы можем оперировать только двумя значениями то мы вынуждены использовать то что называется двоичной системой счисления.
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 шестнадцатиричной системе счисления которая кстати тоже часто используется в компьютерной техники 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 битам 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 схемотехнике 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.

Экран	Слова
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.
Слайд	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 заграница 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.
Слайд	Мы иногда будем возвращаться к бинарным представлениям и разным системам счисления, что называется для общего развития.
таблица «Основные типы данных»	целочисленных типов аж 4 и они занимают 1,2,4,8 байт соответственно. про чар, 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 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

Экран	Слова
byte b = 120; byte b1 = 200;	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? если же мы пишем число которое больше типа слева и соответственно поместиться не может, среда разработки нам выдает сообщение о том что произойдёт переполнение и вообще невозможно положить так много в такой маленький контейнер. А в предупреждении компилятора мы видим, что ожидался байт, а даём мы инт.
лайвкод в котором нужно показать попытку присвоения лонга, показать предупреждения среды	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? поговорм буквально через несколько минут, пока просто запомним, что это происходит
`long l0 = 3_000_000_000L; float f = 0.123f;`	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.

Экран	Слова
Слайд	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. 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 особенности их хранения - преступление, поэтому, снова немного отвлечёмся на общее развитие, несколько минут поговорим не о джаве, а о компьютерах в целом.
много хорошо и подробно, но на С https://habr.com/ru/post/112953/ 	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.
возьмём число +0,5	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.

Экран	Слова
а что если -0,15625	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
$2^1 = 2$ $2^0 = 1.0$ $2^{-1} = 0.5$ $2^{-2} = 0.25$ $2^{-3} = 0.125$ $2^{-4} = 0.0625$ $2^{-5} = 0.03125$ $2^{-6} = 0.015625$ $2^{-7} = 0.0078125$ $2^{-8} = 0.00390625$	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
$(-1)^1 \times 2^{(124-127)} \times (1 + \frac{2097152}{2^{23}}) = 1,15652$ $(-1)^1 \times 1,01e-3 =$ $1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} = 1 \times 0,125 + 0 \times 0,0625 + 1 \times 0,03125 = 0,125 + 0,03125 = 0,15625$	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
Особенности 	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 NaN != 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

Экран	Слова
Проблемы 	у чисел с плавающей запятой могут иногда встречаться и проблемы в вычислениях, пример на слайде чрезвычайно грубый, но при работе, например, со статысячными или миллионными долями во флоуте, с такой проблемой вполне можно столкнуться. порядок выполнения действий может влиять на результат выполнения этих действий, что противоречит математике
таблица «Основные типы данных»	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 тру и фолс, тут я вам ничего нового не скажу, на собесах иногда спрашивают сколько места он занимает, в Java объём хранения не определён и зависит от конкретной жвм, обычно считают, что это байт, несмотря на то что хватит 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
таблица UTF-8	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
Слайд	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 стринг. Поскольку типизация строгая то мы не можем записывать в чары строки а в строки числа

Экран	Слова
Слайд	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.
преобразование типов	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.
Слайд	
константность	
Слайд	
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.
Слайд	Самым простым из ссылочных типов является массив фактически массив выведен на уровень языка и не имеет специального ключевого слово как названия хотя если копнуть гораздо глубже то можно увидеть что у него есть внутреннее название слово эррэй с большой буквы обрамлённое двумя символ 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	