Специализация: данные и функции

Иван Игоревич Овчинников

GeekBrains. Java Core.

2022

Перейдём к интересному: что можно хранить в джаве, как оно там хранится, и как этим манипулировать

В предыдущих сериях

- Краткая история (причины возникновения);
- инструментарий, выбор версии;
- CLI;
- структура проекта;
- документирование;
- некоторые интересные способы сборки проектов.

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).

Типы, преобразование типов

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.

Типы, преобразование типов

рисунок перфокарты

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.

Типы, преобразование типов

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.

Типы, преобразование типов

таблица из методички «Основные типы данных в языке Java»

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 присвавивание в программировании - это не тоже самое, что математическое равенство, а полноценная операция. все присваивания всегда происходят справа налево, то есть сначала вычисляется правая часть, а потом результат вычислений присваивается левой. Именно поэтому в левой части не может быть никакиз вычислений.

Примитивных 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».

Типы, преобразование типов

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/)

(далее, возможно, 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»).

Типы, преобразование типов

целые числа

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:

1. инт - это самый часто используемый тип, если сомневаетесь, какой использовать, используйте инт
2. 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 чаще всего было бы достаточно шорта, например, в циклах, итерирующихся по массивам, или во временных хранениях значений, скажем, возраста человека, но всё равно все по привычке используют инт.
далее - лайвкод в котором нужно показать присвоение к байту и попытку присвоения лонга, показать предупреждения среды.
как мы видим, к маленькому байту вполне успешно присваивается инт. получается, обманул, сказав, что все числа это инты? давайте посмотрим на следующий пример - попытку присвоить значение 5 млрд переменной типа лонг. помним, что в лонге можно хранить очень большие числа, но среда показывает ошибку, 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.

Типы, преобразование типов

немного о хранении чисел с плавающей точкой

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.

Типы, преобразование типов

таблица из методички «Основные типы данных в языке Java»

Что ещё важного мы видив в этой таблице? шесть из восьми примитивных типов могут иметь как положительные, так и отрицательные значения

Типы, преобразование типов

Антипаттерн «магические числа»

кусок Петренко, спасибо ему за идею

В прошлом примере мы использовали антипаттерн - плохой стиль для написания кода. Число 18 используется в коде коде без пояснений. Такой антипаттерн называется ”магическое число”. Рекомендуется помещать числа в константы, которые храняться в начале файла. ADULT = 18 age = float(input(’Ваш возраст: ’)) how_old = age - ADULT print(how_old, ”лет назад ты стал совершеннолетним”)

Плюсом такого подхода является возможность легко корректировать большие проекты. Представьте, что в вашем коде несколько тысяч строк, а число 18 использовалось несколько десятков раз. При развертывании проекта в стране, где совершеннолетием считается 21 год вы будете перечитывать весь код в поисках магических ”18”и править их на ”21”. В случае с константой изменить число нужно в одном месте. Дополнительный сложности могут возникнуть, если в коде будет 18 как возраст совершеннолетия и 18 как коэффициент для рассчёт чего-либо. Теперь править кода ещё сложнее, ведь возраст изменился, а коэффициент -нет. В случае с сохранением значений в константы мы снова меняем число в одном месте.