

1. Специализация: данные и функции

1.1. В предыдущем разделе

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

1.2. В этом разделе

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).

1.3. Данные

1.3.1. Понятие типов

Хранение данных в Java осуществляется привычным для программиста образом: в переменных и константах.

Относительно типизации языки программирования бывают типизированными и нетипизированными (бестиповыми). Нетипизированные языки не представляют большого интереса в современном программировании.

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 не делятся на типы. Работа с ними часто труднее, при этом часто бестиповые языки работают быстрее типизированных, но описывать с их помощью большие проекты со сложными взаимосвязями довольно утомительно.



Java является языком со **строгой** (также можно встретить термин «**сильной**») **явной статической** типизацией.

- Статическая - у каждой переменной должен быть тип, и этот тип изменить нельзя. Этому свойству противопоставляется динамическая типизация;
- Явная - при создании переменной ей обязательно необходимо присвоить какой-то тип, явно написав это в коде. В более поздних версиях языка (с 9й) стало возможным инициализировать переменные типа `var`, обозначающий нужный тип тогда, когда его возможно однозначно вывести из значения справа. Бывают языки с неявной типизацией, например, Python;

— Строгая(сильная) - невозможно смешивать разнотипные данные. С другой стороны, существует JavaScript, в котором запись `2 + true` выдаст результат `3`.

1.3.2. Антипаттерн «магические числа»

Почти во всех примерах, которые используются для обучения, можно увидеть так называемый антипаттерн - плохой стиль для написания кода. Числа, которые находятся справа от оператора присваивания используются в коде без пояснений. Такой антипаттерн называется «магическое число». Магическое, потому что непонятно, что это за число, почему это число именно такое и что будет, если это число изменить.

Так лучше не делать. Заранее нужно сказать, что рекомендуется помещать все числа в коде в именованные константы, которые хранятся в начале файла. Плюсом такого подхода является возможность легко корректировать значения переменных в достаточно больших проектах.

Например, 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 исправить их на «`21`». В этом вопросе будет также важно не запутаться, действительно ли это `18`, которые означают совершеннолетие, а не количество карманов в жилетке Анатолия Вассермана¹.

В случае с константой изменить число нужно в одном месте.

1.4. Примитивные типы данных

Все данные в Java делятся на две основные категории: примитивные и ссылочные. Таблица 2 демонстрирует все восемь примитивных типов языка и их размерности. Чтобы отправить на хранение какие-то данные используется оператор присваивания. 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.

Шесть из восьми типов имеет диапазон значений, а значит основное их отличие в объёме занимаемой памяти. У `double` и `float` тоже есть диапазоны, но они заключаются в точности представления дробной части. Диапазоны означают, что если попытаться положить в переменную меньшего типа большее значение, произойдёт «переполнение переменной».

1.4.1. Переполнение целочисленных переменных

Чем именно чревато переполнение переменной легче показать на примере (по ссылке - расследование крушения ракеты из-за переполнения переменной)



Переполнение переменных не распознаётся компилятором.

¹мы то знаем, что их `26`

Тип	Пояснение	Диапазон
byte	Самый маленький из адресуемых типов, 8 бит, знаковый	[-128, +127]
short	Тип короткого целого числа, 16 бит, знаковый	[-32 768, +32 767]
char	Целочисленный тип для хранения символов в кодировке UTF-8, 16 бит, беззнаковый	[0, +65 535]
int	Основной тип целого числа, 32 бита, знаковый	[-2 147 483 648, +2 147 483 647]
long	Тип длинного целого числа, 64 бита, знаковый	[-9 223 372 036 854 775 808, +9 223 372 036 854 775 807]
float	Тип вещественного числа с плавающей запятой (одинарной точности, 32 бита)	
double	Тип вещественного числа с плавающей запятой (двойной точности, 64 бита)	
boolean	Логический тип данных	true, false

Таблица 1: Основные типы данных в языке Java

Если создать переменную типа byte, диапазон которого от [-128, +127], и присвоить этой переменной значение 200 произойдёт переполнение, как если попытаться влить пакет молока в напёрсток.



Переполнение переменной - это ситуация, в которой происходит попытка положить большее значение в переменную меньшего типа.

Важным вопросом при переполнении остаётся следующий: какое в переполненной переменной останется значение? Максимальное, 127? $200 - 127 = 73$? Какой-то мусор? Каждый язык, а зачастую и разные компиляторы одного языка ведут себя в этом вопросе по-разному.



В современном мире гигагерцев и терабайтов почти никто не пользуется маленькими типами, но именно из-за этого ошибки переполнения переменных становятся опаснее испанской инквизиции.



1.4.2. Задания для самопроверки

1. длдлдл

1.4.3. Бинарное (битовое) представление данных

После разговора о переполнении, нельзя не сказать о том, что именно переполняется. Далее будут представлены сведения которые касаются не только языка Java но и любого другого языка программирования. Эти сведения помогут разобраться в деталях того как хранится значение переменной в программе и как, в целом, происходит работа компьютерной техники.



Все современные компьютеры, так или иначе работают от электричества и являются примитивными по своей 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.

Все данные в любой программе - это единицы и нули. Данные в программе на Java не исключение, удобнее всего это явление рассматривать на примере примитивных данных. Поскольку в компьютере можно оперировать только двумя значениями то естественным образом используется двоичная система счисления.

Десятичное	Двоичное	Восьмеричное	Шестнадцатеричное
00	00000	00	0x00
01	00001	01	0x01
02	00010	02	0x02
03	00011	03	0x03
04	00100	04	0x04
05	00101	05	0x05
06	00110	06	0x06
07	00111	07	0x07
08	01000	10	0x08
09	01001	11	0x09
10	01010	12	0x0a
11	01011	13	0x0b
12	01100	14	0x0c
13	01101	15	0x0d
14	01110	16	0x0e
15	01111	17	0x0f
16	10000	20	0x10

Таблица 2: Представления чисел

Двоичная система счисления это система счисления с основанием два. Существуют и другие системы счисления раньше была очень популярна восьмеричная систему сейчас она отходит на второй план полностью уступая свое место шестнадцатеричной системе счисления которая кстати тоже часто используется в компьютерной техники но сейчас не об этом.

1.4.4. Задания для самопроверки

1. длдлдл

1.4.5.

1.5. Базовый функционал языка

1.5.1. Математические операторы

1.5.2. Условия

1.5.3. Циклы

1.5.4. Бинарные арифметические операторы

;

1.6. Функции

Задания к семинару

- Написать как можно больше вариантов функции инвертирования массива единиц и нулей за 15 минут (без ветвлений любого рода);
- Сравнить без условий две даты, представленные в виде трёх чисел гггг-мм-дд;