ТИПЫ ДАННЫХ И ОПЕРАЦИИ

Методические указания к лабораторной работе

***Цель работы**:* Изучить типы данных языка Java и уметь работать со значениями разных типов.

Теоретическая справка

Примитивные типы данных и операции

Все типы исходных данных, встроенные в язык Java, делятся на две группы: *примитивные типы* (primitive types) и *ссылочные типы* (reference types).

Ссылочные типы делятся на *массивы* (arrays), *классы* (classes) и *интерфейсы* (interfaces).

Примитивных типов всего восемь. Их можно разделить на *логический* (иногда говорят *булев)* тип boolean и *числовые* (numeric).

К числовым типам относятся *целые* (integral [Название "integral" не является устоявшимся термином. Так названа категория целых типов данных в книге The Java Language Specification, Second Edition. James Gosling, Bill Joy, Guy Steele, Gilad Bracha (см. введение). — Ред.]) и *вещественные* (floating-point) типы.

Целых типов пять: byte , short , int , long , char .

Символы можно использовать везде, где используется тип int , поэтому JLS причисляет их к целым типам. Например, их можно использовать в арифметических вычислениях, скажем, можно написать 2 + 'ж' , к двойке будет прибавляться кодировка Unicode '\u04i6' буквы 'ж' . В десятичной форме это число 1046 и в результате сложения получим 1048.

Напомним, что в записи 2 + "Ж" плюс понимается как сцепление строк, двойка будет преобразована в строку, в результате получится строка "2ж" .

Вещественных типов два: float и double .

Поскольку по имени переменной невозможно определить ее тип, все переменные обязательно должны быть описаны перед их использованием. Описание заключается в том, что записывается имя типа, затем, через пробел, список имен переменных, разделенных запятой. Для всех или некоторых переменных можно указать начальные значения после знака равенства, которыми могут служить любые константные выражения того же типа. Описание каждого типа завершается точкой с запятой. В программе может быть сколько угодно описаний каждого типа.

Разберем каждый тип подробнее.

Логический тип

Значения логического типа boolean возникают в результате различных сравнений, вроде 2 > 3, и используются, главным образом, в условных операторах и операторах циклов. Логических значении всего два: true (истина) и false (ложь). Описание переменных этого типа выглядит так:

```
boolean b = true, bb = false, bool2;
```

Над логическими данными можно выполнять операции присваивания, например, bool2 = true , в том числе и составные с логическими операциями; сравнение на равенство b == bb и на неравенство b != bb , а также логические операции.

Логические операции:

- отрицание (NOT) ! (обозначается восклицательным знаком);
- конъюнкция (AND) & (амперсанд);
- дизъюнкция (OR) | (вертикальная черта);
- исключающее ИЛИ (XOR) ^ (каре).

Они выполняются над логическими данными, их результатом будет тоже логическое значение true или false . Про них можно ничего не знать, кроме того, что представлено в табл. 1.1.

Таблица 1.1. Логические операции

b1	b2	!b1	b1&b2	b1|b2	b1^b2
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Целые типы

Спецификация языка Java, JLS, определяет разрядность (количество байтов, выделяемых для хранения значений типа в оперативной памяти) и диапазон значений каждого типа. Для целых типов они приведены в табл. 1.2.

Таблица 1.2. Целые типы

Тип	Разрядность (байт)	Диапазон
byte	1	от -128 до 127

short	2	от -32768 до 32767
int	4	от -2147483648 до 2147483647
long	8	от -9223372036854775808 до 9223372036854775807
char	2	от '\u0000' до '\uFFFF' , в десятичной форме от 0 до 65535

Впрочем, для Java разрядность не столь важна, на некоторых компьютерах она может отличаться от указанной в таблице, а вот диапазон значений должен выдерживаться неукоснительно.

Хотя тип char занимает два байта, в арифметических вычислениях он участвует как тип int , ему выделяется 4 байта, два старших байта заполняются нулями.

Примеры определения переменных целых типов:

```
byte b1 = 50, b2 = -99, b3;

short det = 0, ind = 1;

int i = -100, j = 100, k = 9999;

long big = 50, veryBig = 2147483648L;

char c1 = 'A', c2 = '?', newLine = '\n';
```

Целые типы хранятся в двоичном виде с дополнительным кодом. Последнее означает, что для отрицательных чисел хранится не их двоичное представление, а *дополнительный код* этого двоичного представления.

Дополнительный же код получается так: в двоичном представлении все нули меняются на единицы, а единицы на нули, после чего к результату прибавляется единица, разумеется, в двоичной арифметике.

Например, значение 50 переменной b1 , определенной выше, будет храниться в одном байте с содержимым 00110010 , а значение -99 переменной b2 — в байте с содержимым, которое вычисляем так: число 99 переводим в двоичную форму, получая 01100011 , меняем единицы и нули, получая 10011100 , и прибавляем единицу, получив окончательно байт с содержимым 10011101 .

Смысл всех этих сложностей в том, что сложение числа с его дополнительным кодом в двоичной арифметике даст в результате нуль, старший бит просто теряется. Это означает, что в такой странной арифметике дополнительный код числа является противоположным к нему числом, числом с обратным знаком. А это, в свою очередь, означает, что вместо того, чтобы вычесть из числа A число B, можно к A прибавить дополнительный код числа B. Таким 'образом, операция вычитания исключается из набора машинных операций.

Над целыми типами можно производить массу операций. Их набор восходит к языку C, он оказался удобным и кочует из языка в язык почти без изменений.

Все операции, которые производятся над целыми числами, можно разделить на следующие группы.

Арифметические операции

К арифметическим операциям относятся:

- сложение + (плюс);
- вычитание - (дефис);
- умножение * (звездочка);
- деление / (наклонная черта — слэш);
- взятие остатка от деления (деление по модулю) % (процент);
- инкремент (увеличение на единицу) ++ ;
- декремент (уменьшение на единицу) --

Между сдвоенными плюсами и минусами нельзя оставлять пробелы. Сложение, вычитание и умножение целых значений выполняются как обычно, а вот деление целых значений в результате дает опять целое (так называемое *"целое деление"),* например, 5/2 даст в результате 2 , а не 2.5 , а 5/(-3) даст -1 . Дробная часть попросту отбрасывается, происходит усечение частного. Это поначалу обескураживает, но потом оказывается удобным для усечения чисел.

Замечание

В Java принято целочисленное деление.

Это странное для математики правило естественно для программирования: если оба операнда имеют один и тот же тип, то и результат имеет тот же тип. Достаточно написать 5/2.0 или 5.0/2 или 5.0/2.0 и получим 2.5 как результат деления вещественных чисел.

Операция *деление по модулю* определяется так: a % b = a - (a / b) * b ; например, 5%2 даст в результате 1 , а 5% (-3) даст, 2 , т.к. 5 = (-3) * (-1) + 2 , но (-5)%3 даст -2 , поскольку -5 = 3 * (-1) - 2 .

Операции *инкремент* и *декремент* означают увеличение или уменьшение значения переменной на единицу и применяются только к переменным, но не к константам или выражениям, нельзя написать 5++ или (a + b)++ .

Например, после приведенных выше описаний i++ даст -99 , а j—- даст 99 .

Интересно, что эти операции можно записать и перед переменной: ++i , — j . Разница проявится только в выражениях: при первой форме записи *(постфиксной)* в выражении участвует старое значение переменной и только потом происходит увеличение или уменьшение ее значения. При второй форме записи *(префиксной)* сначала изменится переменная и ее новое значение будет участвовать в выражении.

Например, после приведенных выше описаний, (k++) + 5 даст в результате 10004 , а переменная k примет значение 10000 . Но в той же исходной ситуации (++k) + 5 даст 10005 , а переменная k станет равной 10000 .

Приведение типов

Результат арифметической операции имеет тип int, кроме того случая, когда один из операндов типа long . В этом случае результат будет типа long .

Перед выполнением арифметической операции всегда происходит *повышение* (promotion) типов byte , short , char . Они преобразуются в тип int , а может быть, и в тип long , если другой операнд типа long . Операнд типа int повышается до типа long , если другой операнд типа long . Конечно, числовое значение операнда при этом не меняется.

Можно выполнить явное приведение типа. Например, *сужение* (narrowing) типа int до типа short осуществляется операцией явного приведения, которая записывается перед приводимым значением в виде имени типа в скобках:

```
short k = (short)(b1 + b2) ;
```

Операции сравнения

В языке Java шесть обычных операций сравнения целых чисел по величине:

- больше > ;
- меньше < ;
- больше или равно >= ;
- меньше или равно <= ;
- равно == ;
- не равно != .

Сдвоенные символы записываются без пробелов, их нельзя переставлять местами, запись => будет неверной.

Результат сравнения — логическое значение: true , в результате, например, сравнения 3 != 5 ; или false , например, в результате сравнения 3 == 5 .

Для записи сложных сравнений следует привлекать логические.операции. Например, в вычислениях часто приходится делать проверки вида a < x < b . Подобная запись на языке Java приведет к сообщению об ошибке, поскольку первое сравнение, a < x , даст true или false , а Java не знает, больше это, чем b , или меньше. В данном случае следует написать выражение (a < x) && (x < b) , причем здесь скобки можно опустить, написать просто a < x && x < b , но об этом немного позднее.

Побитовые операции

Иногда приходится изменять значения отдельных битов в целых данных. Это выполняется с помощью побитовых (bitwise) операций путем наложения маски. В языке Java есть четыре побитовые операции:

- дополнение (complement) ~ (тильда);
- побитовая конъюнкция (bitwise AND) & ;
- побитовая дизъюнкция (bitwise OR) | ;
- побитовое исключающее ИЛИ (bitwise XOR) ^ .

Они выполняются поразрядно, после того как оба операнда будут приведены к одному типу int или long , так же как и для арифметических операций, а значит, и к одной разрядности. Операции над каждой парой битов выполняются согласно табл. 1.3.

Таблица 1.3. Побитовые операции

n1	n2	~n1	n1 & n2	n1 | n2	n1 ^ n2
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	0	0	0

В нашем примере b1 == 50 , двоичное представление 00110010, b2 == -99 , двоичное представление 10011101 . Перед операцией происходит повышение до типа int . Получаем представления из 32-х разрядов для b1 — 0...00110010 , для b2 — 1...l0011101 . В результате побитовых операций получаем:

- ~b2 == 98 , двоичное представление 0...01100010 ;
- b1 & b2 == 16 , двоичное представление 0...00010000 ;
- b1 | b2 == -65 , двоичное представление 1...10111111 ;
- b1 ^ b2 == -81 , двоичное представление 1...10101111 .

Двоичное представление каждого результата занимает 32 бита.

Заметьте, что дополнение ~x всегда эквивалентно (-x)-1 .

Сдвиги

В языке Java есть три операции сдвига двоичных разрядов:

- сдвиг влево <<;
- сдвиг вправо >>;
- беззнаковый сдвиг вправо >>>.

Эти операции своеобразны тем, что левый и правый операнды в них имеют разный смысл. Слева стоит значение целого типа, а правая часть показывает, на сколько двоичных разрядов сдвигается значение, стоящее в левой части.

Например, операция b1<< 2 сдвинет влево на 2 разряда предварительно повышенное значение 0...00110010 переменной b1, что даст в результате 0...011001000, десятичное 200. Освободившиеся справа разряды заполняются нулями, левые разряды, находящиеся за 32-м битом, теряются.

Операция b2 << 2 сдвинет повышенное значение 1...10011101 на два разряда влево. В результате получим 1...1001110100, десятичное значение —396.

Заметьте, что сдвиг влево на n разрядов эквивалентен умножению числа на 2 в степени n.

Операция b1 >> 2 даст в результате 0...00001100, десятичное 12, а b2 >> 2 — результат 1..11100111, десятичное -25, т. е. слева распространяется старший бит, правые биты теряются. Это так называемый *арифметический сдвиг.*

Операция беззнакового сдвига во всех случаях ставит слева на освободившиеся места нули, осуществляя *логический сдвиг. Но* вследствие предварительного повышения это имеет эффект только для нескольких старших разрядов отрицательных чисел. Так, b2 >>> 2 имеет результатом 001...100111, десятичное число 1 073 741 799.

Вещественные типы

Вещественных типов в Java два: float и double. Они характеризуются разрядностью, диапазоном значений и точностью представления, отвечающим стандарту IEEE 754-1985 с некоторыми изменениями. К обычным вещественным числам добавляются еще три значения:

1. Положительная бесконечность, выражаемая константой POSITIVE_INFINITY и возникающая при переполнении положительного значения, например, в результате операции умножения 3.0*6e307.

2. Отрицательная бесконечность NEGATIVE_INFINITY .

3. "Не число", записываемое константой NaN (Not a Number) и возникающее при делении вещественного числа на нуль или умножении нуля на бесконечность.

Кроме того, стандарт различает положительный и отрицательный нуль, возникающий при делении на бесконечность соответствующего знака, хотя сравнение o.o == -o.o дает true.

Операции с бесконечностями выполняются по обычным математическим правилам.

Во всем остальном вещественные типы — это обычные, вещественные значения, к которым применимы все арифметические операции и сравнения, перечисленные для целых типов. Характеристики вещественных типов приведены в табл. 1.4.

В языке Java взятие остатка*от деления %, инкремент ++ и декремент — применяются и к вещественным типам.

Таблица 1.4. Вещественные типы

Тип	Разрядность	Диапазон	Точность
float	4	3,4e-38 < |x| < 3,4e38	7—8 цифр
double	8	1,7e-308<|x|<1,7e308	17 цифр

Примеры определения вещественных типов:

```
float x = 0.001, y = -34.789;

double 21 = -16.2305, z2;
```

Поскольку к вещественным типам применимы все арифметические операции и сравнения, целые и вещественные значения можно смешивать в операциях. При этом правило приведения типов дополняется такими условиями:

- если в операции один операнд имеет тип double, то и другой приводится к типу double;
- если один операнд имеет тип float, то и другой приводится к типу float;
- в противном случае действует правило приведения целых значений.

Операции присваивания

Простая операция присваивания (simple assignment operator) записывается знаком равенства =, слева от которого стоит переменная, а справа выражение, совместимое с типом переменной:

```
x = 3.5, y = 2 * (x - 0.567) / (x + 2), b = x < y, bb = x >= y && b.
```

Операция присваивания действует так: выражение, стоящее после знака равенства, вычисляется и приводится к типу переменной, стоящей слева от знака равенства. Результатом операции будет приведенное значение правой части.

Операция присваивания имеет еще одно, побочное, действие: переменная, стоящая слева, получает приведенное значение правой части, старое ее значение теряется.

В операции присваивания левая и правая части неравноправны, нельзя написать 3.5 = x. После операции x = y изменится переменная x, став равной y, а после y = x изменится y.

Кроме простой операции присваивания есть еще 11 *составных* операций присваивания (compound assignment operators):

+=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>= ; >>>=.

Символы записываются без пробелов, нельзя переставлять их местами.

Все составные операции присваивания действуют по одной схеме:

x *op*= *a* эквивалентно x = *(тип* x), т. е. (x op a).

Перед присваиванием, при необходимости, автоматически производится приведение типа. Поэтому:

```
byte b = 1;

b = b + 10; // Ошибка!

b += 10; // Правильно!
```

Перед сложением b + 50 происходит повышение b до типа int, результат сложения тоже будет типа int и, в первом случае, не может быть присвоен переменной b без явного приведения типа. Во втором случае перед присваиванием произойдет сужение результата сложения до типа byte.

Условная операция

Эта своеобразная операция имеет три операнда. Вначале записывается произвольное логическое выражение, т. е. имеющее в результате true или false, затем знак вопроса, потом два произвольных выражения, разделенных двоеточием, например,

```
x < 0 ? 0 : x

x > y ? x  — y : x + y
```

Условная операция выполняется так. Сначала вычисляется логическое выражение. Если получилось значение true, то вычисляется первое выражение после вопросительного знака ? и его значение будет результатом всей операции. Последнее выражение при этом не вычисляется. Если же получилось значение false, то вычисляется только последнее выражение, его значение будет результатом операции.

Это позволяет написать n == o ? да : m / n не опасаясь деления на нуль. Условная операция поначалу кажется странной, но она очень удобна для записи небольших разветвлений.

Выражения

Из констант и переменных, операций над ними, вызовов методов и скобок составляются *выражения* (expressions). Разумеется, все элементы выражения должны быть совместимы, нельзя написать, например, 2 + true. При вычислении выражения выполняются четыре правила:

1. Операции одного приоритета вычисляются слева направо: x + y + z вычисляется как (x + y) + z. Исключение: операции присваивания вычисляются справа налево: x = y = z вычисляется как x = (y = z).

2. Левый операнд вычисляется раньше правого.

3. Операнды полностью вычисляются перед выполнением операции.

4. Перед выполнением составной операции присваивания значение левой части сохраняется для использования в правой части.

Следующие примеры показывает особенности применения первых трех правил. Пусть

int a = 3, b = 5;

Тогда результатом выражения b + (b = 3) будет число 8; но результатом выражения (b = 3) + b будет число 6. Выражение b += (b = 3) даст в результате 8, потому что вычисляется как первое из приведенных выше выражений.

Четвертое правило можно продемонстрировать так. При тех же определениях a и b в результате вычисления выражения b += a += b += 7 получим 20. Хотя операции присваивания выполняются справа налево и после первой, правой, операции значение b становится равным 12, но в последнем, левом, присваивании участвует старое значение b, равное 5. А в результате двух последовательных вычислений a += b += 7; b += a; получим 27, поскольку во втором выражении участвует уже новое значение переменной b, равное 12.

Выражения могут иметь сложный и запутанный вид. В таких случаях возникает вопрос о приоритете операций, о том, какие операции будут выполнены в первую очередь. Естественно, умножение и деление производится раньше сложения и вычитания. Остальные правила перечислены ниже.

Порядок вычисления выражения всегда можно отрегулировать скобками, их можно вставить сколько угодно. Но здесь важно соблюдать "золотую середину". При большом количестве скобок снижается наглядность выражения и легко ошибиться в расстановке скобок. Если выражение со скобками корректно, то компилятор может отследить только парность скобок, но не правильность их расстановки.

Приоритет операций

Операции перечислены в порядке убывания приоритета. Операции на одной строке имеют одинаковый приоритет.

1. Постфиксные операции ++ и —.

2. Префиксные операции ++ и —, дополнение ~ и отрицание !.

3. Приведение типа (тип).

4. Умножение *, деление / и взятие остатка %.

5. Сложение + и вычитание -.

6. Сдвиги <<, >>, >>>.

7. Сравнения >, <, >=, <=.

8. Сравнения ==, !=.

9. Побитовая конъюнкция &.

10. Побитовое исключающее ИЛИ ^.

11. Побитовая дизъюнкция | .

12. Конъюнкция &&.

13. Дизъюнкция | | .

14. Условная операция ?: .

15. Присваивания =, +=, -=, *=, /=, %=, &=, ^=, |=, <<, >>, >>>.

Здесь перечислены не все операции языка Java, список будет дополняться по мере изучения новых операций.

Задание на работу

Объявить в методе main класса Simple столько же переменных типа char, сколько разных букв необходимо для написания Ваших имени и фамилии, и инициализировать их соответствующими символами при объявлении. Гарантировать невозможность изменения этих переменных в дальнейшем.

Объявить две переменные типа String и две переменные целого типа (выбрать конкретный целый тип самостоятельно, исходя из требуемого диапазона допустимых значений). Значения переменных типа String нужно получить конкатенацией char переменных (это должны быть фамилия и имя соответственно), значения переменных целого типа являются суммами кодов символов, составляющих имя и фамилию соответственно. При подсчёте этих сумм нельзя использовать операцию явного приведения типа.

Объявить две переменные вещественного типа и присвоить им значения полученных сумм.

Вывести на экран:

- значения строковых переменных,

- значения переменных целого типа,
- значения вещественных переменных,
- результат деления суммы, соответствующей фамилии, на сумму, соответсвующую имени, а также остаток от этого деления, для целых и вещественных переменных.

Для вывода результатов использовать метод System.out.println(), обязательно добавить комментарии ко всем выводимым числовым значениям.

Нижеприведённый шаблон объявления класса Simple необходимо дополнить нужным кодом и сохранить в файле Simple.java:

```
public class Simple {
    public static void main(String[] args) {

            //Some code should be here

            System.out.println("output");
    }
}
```

Для компиляции программы нужно ввести следующую команду в командной строке:

<jdk\bin>javac Simple.java

где <jdk\bin> - путь к файлу javac.exe.

Выполнить программу можно следующей командой:

java -cp <path> Simple

где <path> - путь, по которому располагается файл Simple.class. Если в именах папок есть пробелы, путь нужно взять в кавычки.

Отчёт по работе

Отчёт по работе должен содержать титульный лист, название работы, цель работы, структурированный код программы и результат её работы.