Программирование на языке Java. Типы

Картузов А.В.

В этой главе вы познакомитесь со всеми основными типами языка Java и увидите, как надо объявлять переменные, присваивать им значения и использовать выражения со смешанными типами. В данной главе мы и обсудим простые типы языка Java, оставив сложные типы до главы 7.

1. Простые типы

Простые типы в Java не являются объектно-ориентированными, они аналогичны простым типам большинства традиционных языков программирования. В Java имеется восемь простых типов:—byte, short, int, long, char, float, double и boolean. Их можно разделить на четыре группы:

- Целые. К ним относятся типы byte, short, int и long. Эти типы предназначены для целых чисел со знаком.
- Типы с плавающей точкой—float и double. Они служат для представления чисел, имеющих дробную часть.
- Символьный тип char. Этот тип предназначен для представления элементов из таблицы символов, например, букв или цифр.
- Логический тип boolean. Это специальный тип, используемый для представления логических величин.

В Java, в отличие от некоторых других языков, отсутствует автоматическое приведение типов. Несовпадение типов приводит не к предупреждению при трансляции, а к сообщению об ошибке. Для каждого типа строго определены наборы допустимых значений и разрешенных операций.

2. Целые числа

В языке Java понятие беззнаковых чисел отсутствует. Все числовые типы этого языка—знаковые. Например, если значение переменной типа byte равно в шестнадцатиричном виде 0x80, то это—число -1.

Замечание:

Единственная реальная причина использования беззнаковых чисел—это использование иных, по сравнению со знаковыми числами, правил манипуляций с битами при выполнении операций сдвига. Пусть, например, требуется сдвинуть вправо битовый массив mask, хранящийся в целой переменной и избежать при этом расширения знакового разряда, заполняющего старшие биты единицами. Стандартный способ выполнения этой задачи в C—((unsigned) mask) >> 2. В Java для этой цели введен новый оператор беззнакового сдвига вправо. Приведенная выше операция записывается с его помощью в виде mask>>>2. Детально мы обсудим все операторы в следующей главе.

Отсутствие в Java беззнаковых чисел вдвое сокращает количество целых типов. В языке имеется 4 целых типа, занимающих 1, 2, 4 и 8 байтов в памяти. Для каждого типа—byte, short, int и long, есть свои естественные области применения.

2.1. byte

Тип byte—это знаковый 8-битовый тип. Его диапазон—от -128 до 127. Он лучше всего подходит для хранения произвольного потока байтов, загружаемого из сети или из файла.

```
byte b;
byte c = 0x55;
```

Если речь не идет о манипуляциях с битами, использования типа byte, как правило, следует избегать. Для нормальных целых чисел, используемых в качестве счетчиков и в арифметических выражениях, гораздо лучше подходит тип int.

2.2. short

short—это знаковый 16-битовый тип. Его диапазон—от -32768 до 32767. Это, вероятно, наиболее редко используемый в Java тип, поскольку он определен, как тип, в котором старший байт стоит первым.

```
short s;
short t = 0x55aa;
```

Замечание:

Случилось так, что на ЭВМ различных архитектур порядок байтов в слове различается, например, старший байт в двухбайтовом целом short может храниться первым, а может и последним. Первый случай имеет место в архитектурах SPARC и Power PC, второй—для микропроцессоров Intel x86. Переносимость программ Java требует, чтобы целые значения одинаково были представлены на ЭВМ разных архитектур.

2.3. int

Тип int служит для представления 32-битных целых чисел со знаком. Диапазон допустимых для этого типа значений—от -2147483648 до 2147483647. Чаще всего этот тип данных используется для хранения обычных целых чисел со значениями, достигающими двух миллиардов. Этот тип прекрасно подходит для использования при обработке массивов и для счетчиков. В ближайшие годы этот тип будет прекрасно соответствовать машинным словам не только 32-битовых процессоров, но и 64-битовых с поддержкой быстрой конвейеризации для выполнения 32-битного кода в режиме совместимости. Всякий раз, когда в одном выражении фигурируют переменные типов byte, short, int и целые литералы, тип всего выражения перед завершением вычислений приводится к int.

```
int i;
int j = 0x55aa0000;
```

2.4. long

Тип long предназначен для представления 64-битовых чисел со знаком. Его диапазон допустимых значений достаточно велик даже для таких задач, как подсчет числа атомов во вселенной.

```
long m;
long n = 0x55aa000055aa0000;
```

Не надо отождествлять разрядность целочисленного типа с занимаемым им количеством памяти. Исполняющий код Java может использовать для ваших

переменных то количество памяти, которое сочтет нужным, лишь бы только их поведение соответствовало поведению типов, заданных вами. Фактически, нынешняя реализация Java из соображений эффективности хранит переменные типа byte и short в виде 32-битовых значений, поскольку этот размер соответствует машинному слову большинства современных компьютеров (CM—8 бит, 8086—16 бит, 80386/486—32 бит, Pentium—64 бит).

Ниже приведена таблица разрядностей и допустимых диапазонов для различных типов целых чисел.

Имя	Разрядность	Диапазон
long	64	-9, 223, 372, 036, 854, 775, 808.. 9, 223, 372, 036, 854, 775, 807
Int	32	-2, 147, 483, 648.. 2, 147, 483, 647
Short	16	-32, 768.. 32, 767
byte	8	-128.. 127

3. Числа с плавающей точкой

Числа с плавающей точкой, часто называемые в других языках вещественными числами, используются при вычислениях, в которых требуется использование дробной части. В Java реализован стандартный (IEEE-754) набор типов для чисел с плавающей точкой—float и double и операторов для работы с ними. Характеристики этих типов приведены в таблице.

Имя	Разрядность	Диапазон
double	64	1.7e-308.. 1. 7e+ 308
float	32	3.4e-038.. 3. 4e+ 038

3.1. float

В переменных с обычной, или одинарной точностью, объявляемых с помощью ключевого слова float, для хранения вещественного значения используется 32 бита.

```
        float f;
        float f2 = 3. 14F;              // обратите внимание на F,
                                 //т.к. по умолчанию все литералы double
```

3.2. double

В случае двойной точности, задаваемой с помощью ключевого слова double, для хранения значений используется 64 бита. Все трансцендентные математические функции, такие, как sin, cos, sqrt, возвращают результат типа double.

```
        double d;
        double pi = 3. 14159265358979323846;
```

4. Приведение типа

Приведение типов (type casting)—одно из неприятных свойств C++, тем не менее приведение типов сохранено и в языке Java. Иногда возникают ситуации, когда у вас есть величина какого-то определенного типа, а вам нужно ее присвоить переменной другого типа. Для некоторых типов это можно проделать и без приведения типа, в таких случаях говорят об автоматическом преобразовании типов. В Java автоматическое преобразование возможно только в том случае, когда точности представления чисел переменной-приемника достаточно для хранения исходного значения. Такое преобразование происходит, например, при занесении литеральной константы или значения переменной типа byte или short в переменную типа int. Это называется расширением (widening) или повышением (promotion), поскольку тип меньшей разрядности расширяется (повышается) до большего совместимого типа. Размера типа int всегда достаточно для хранения чисел из диапазона, допустимого для типа byte, поэтому в подобных ситуациях оператора явного приведения типа не требуется. Обратное в большинстве случаев неверно, поэтому для занесения значения типа int в переменную типа byte необходимо использовать оператор приведения типа. Эту процедуру иногда называют сужением (narrowing), поскольку вы явно сообщаете транслятору, что величину необходимо преобразовать, чтобы она уместилась в переменную нужного вам типа. Для приведения величины к определенному типу перед ней нужно указать этот тип, заключенный в круглые скобки. В приведенном ниже фрагменте кода демонстрируется приведение типа источника (переменной типа int) к типу приемника (переменной типа byte). Если бы при такой операции целое

значение выходило за границы допустимого для типа byte диапазона, оно было бы уменьшено путем деления по модулю на допустимый для byte диапазон (результат деления по модулю на число—это остаток от деления на это число).

```
int a = 100;
byte b = (byte) a;
```

4.1. Автоматическое преобразование типов в выражениях

Когда вы вычисляете значение выражения, точность, требуемая для хранения промежуточных результатов, зачастую должна быть выше, чем требуется для представления окончательного результата.

```
byte a = 40;
byte b = 50;
byte c = 100;
int d = a* b / c;
```

Результат промежуточного выражения (a* b) вполне может выйти за диапазон допустимых для типа byte значений. Именно поэтому Java автоматически повышает тип каждой части выражения до типа int, так что для промежуточного результата (a* b) хватает места.

Автоматическое преобразование типа иногда может оказаться причиной неожиданных сообщений транслятора об ошибках. Например, показанный ниже код, хотя и выглядит вполне корректным, приводит к сообщению об ошибке на фазе трансляции. В нем мы пытаемся записать значение 50* 2, которое должно прекрасно уместиться в тип byte, в байтовую переменную. Но из-за автоматического преобразования типа результата в int мы получаем сообщение об ошибке от транслятора—ведь при занесении int в byte может произойти потеря точности.

```
byte b = 50;
b = b* 2;
^ Incompatible type for =. Explicit cast needed to convert int to byte.
```

(Несовместимый тип для =. Необходимо явное преобразование int в byte)

Исправленный текст :

```
        byte b = 50;
        b = (byte) (b* 2);
```

что приводит к занесению в b правильного значения 100.

Если в выражении используются переменные типов byte, short и int, то во избежание переполнения тип всего выражения автоматически повышается до int. Если же в выражении тип хотя бы одной переменной—long, то и тип всего выражения тоже повышается до long. Не забывайте, что все целые литералы, в конце которых не стоит символ L (или 1), имеют тип int.

Если выражение содержит операнды типа float, то и тип всего выражения автоматически повышается до float. Если же хотя бы один из операндов имеет тип double, то тип всего выражения повышается до double. По умолчанию Java рассматривает все литералы с плавающей точкой, как имеющие тип double. Приведенная ниже программа показывает, как повышается тип каждой величины в выражении для достижения соответствия со вторым операндом каждого бинарного оператора.

```
class Promote {
        public static void main (String args []) { byte b = 42;
                char c = 'a';
                short s = 1024;
                int i = 50000;
                float f = 5.67f;
                double d =.1234;
                double result = (f* b) + (i/ c)—(d* s);
                System. out. println ((f* b)+ "+ "+ (i / c)+ "—" + (d* s));
                System. out. println ("result = "+ result);
                }
        }
```

Подвыражение f* b—это число типа float, умноженное на число типа byte. Поэтому его тип автоматически повышается до float. Тип следующего подвыражения i / c (int, деленный на char) повышается до int. Аналогично этому тип подвыражения d* s (double, умноженный на short) повышается до double. На следующем шаге вычислений мы имеем дело с тремя промежуточными результатами типов float, int и double. Сначала при сложении первых двух тип int повышается до float и получается результат

типа float. При вычитании из него значения типа double тип результата повышается до double. Окончательный результат всего выражения—значение типа double.

5. Символы

Поскольку в Java для представления символов в строках используется кодировка Unicode, разрядность типа char в этом языке—16 бит. В нем можно хранить десятки тысяч символов интернационального набора символов Unicode. Диапазон типа char—0..65536. Unicode—это объединение десятков кодировок символов, он включает в себя латинский, греческий, арабский алфавиты, кириллицу и многие другие наборы символов.

```
char c;
char c2 = Oxf132;
char c3 = ' a';
char c4 = '\n';
```

Хотя величины типа char и не используются, как целые числа, вы можете оперировать с ними так, как если бы они были целыми. Это дает вам возможность сложить два символа вместе, или инкрементировать значение символьной переменной. В приведенном ниже фрагменте кода мы, располагая базовым символом, прибавляем к нему целое число, чтобы получить символьное представление нужной нам цифры.

```
int three = 3;
char one = '1';
char four = (char) (three+ one);
```

В результате выполнения этого кода в переменную four заносится символьное представление нужной нам цифры—'4'. Обратите внимание—тип переменной one в приведенном выше выражении повышается до типа int, так что перед занесением результата в переменную four приходится использовать оператор явного приведения типа.

6. Тип boolean

В языке Java имеется простой тип boolean, используемый для хранения логических

значений. Переменные этого типа могут принимать всего два значения—true (истина) и false (ложь). Значения типа boolean возвращаются в качестве результата всеми операторами сравнения, например (a < b)—об этом разговор пойдет в следующей главе. Кроме того, в главе 6 вы узнаете, что boolean—это тип, требуемый всеми условными операторами управления—такими, как if, while, do.

```
boolean done = false;
```

Завершая разговор о простых типах...

Теперь, когда мы познакомились со всеми простыми типами, включая целые и вещественные числа, символы и логические переменные, давайте попробуем собрать всю информацию вместе. В приведенном ниже примере создаются переменные каждого из простых типов и выводятся значения этих переменных.

```
class SimpleTypes {
        public static void main(String args []) {
                byte b = 0x55;
                short s = 0x55ff;
                int i = 1000000;
                long l = 0xffffffffL;
                char c = ' a' ;
                float f = .25f;
                double d = .00001234;
                boolean bool = true;
                System.out.println("byte b = " + b);
                System.out.println("short s = " +s);
                System.out.println("int i = " + i);
                System.out.println("long l = " + l);
                System.out.println("char c = " + c);
                System.out.println("float f = " + f);
                System.out.println("double d = " + d);
                System.out.println("boolean bool = " + bool);
                }
        }
```

Запустив эту программу, вы должны получить результат, показанный ниже:

```
C: \> java SimpleTypes
```

```
byte b = 85
short s = 22015
int i = 1000000
long l = 4294967295
char c = a
float f = 0.25
double d = 1.234e-005
boolean bool = true
```

Обратите внимание на то, что целые числа печатаются в десятичном представлении, хотя мы задавали значения некоторых из них в шестнадцатиричном формате. В главе 12 вы узнаете, как можно форматировать выводимые числовые значения.

7. Массивы

Для объявления типа массива используются квадратные скобки. В приведенной ниже строке объявляется переменная month_days, тип которой—"массив целых чисел типа int".

```
int month_days [];
```

Для того, чтобы зарезервировать память под массив, используется специальный оператор new. В приведенной ниже строке кода с помощью оператора new массиву month_days выделяется память для хранения двенадцати целых чисел.

```
month_days = new int [12];
```

Итак, теперь month_days—это ссылка на двенадцать целых чисел. Ниже приведен пример, в котором создается массив, элементы которого содержат число дней в месяцах года (невисокосного).

```
class Array {
        public static void main (String args []) {
                int month_days[];
                month_days = new int[12];
                month_days[0] = 31;
                month_days[1] = 28;
                month_days[2] = 31;
```

```
                month_days[3] = 30;
                month_days[4] = 31;
                month_days[5] = 30;
                month_days[6] = 31;
                month_days[7] = 31;
                month_days[8] = 30;
                month_days[9] = 31;
                month_days[10] = 30;
                month_days[11] = 31;
                System.out.println("April has " +
                        month_days[3] + " days.");
                }
        }
```

При запуске эта программа печатает количество дней в апреле, как это показано ниже. Нумерация элементов массива в Java начинается с нуля, так что число дней в апреле—это month_days [3].

```
C: \> java Array
April has 30 days.
```

Имеется возможность автоматически инициализировать массивы способом, во многом напоминающим инициализацию переменных простых типов. Инициализатор массива представляет собой список разделенных запятыми выражений, заключенный в фигурные скобки. Запятые отделяют друг от друга значения элементов массива. При таком способе создания массив будет содержать ровно столько элементов, сколько требуется для хранения значений, указанных в списке инициализации.

```
class AutoArray {
        public static void main(String args[]) {
                int month_days[] = { 31, 28, 31, 30, 31, 30,
                            31, 31, 30, 31, 30, 31 };
                System.out.println("April has " +
                                        month_days[3] + " days.");
                }
        }
```

В результате работы этой программы, вы получите точно такой же результат, как и от ее более длинной предшественницы.

Java строго следит за тем, чтобы вы случайно не записали или не попытались получить значения, выйдя за границы массива. Если же вы попытаетесь использовать в качестве индексов значения, выходящие за границы массива—отрицательные числа либо числа, которые больше или равны количеству элементов в массиве, то получите сообщение об ошибке времени выполнения. В главе 10 мы подробно расскажем о том, что делать при возникновении подобных ошибок.

7.1. Многомерные массивы

На самом деле, настоящих многомерных массивов в Java не существует. Зато имеются массивы массивов, которые ведут себя подобно многомерным массивам, за исключением нескольких незначительных отличий. Приведенный ниже код создает традиционную матрицу из шестнадцати элементов типа double, каждый из которых инициализируется нулем. Внутренняя реализация этой матрицы—массив массивов double.

```
double matrix [][] = new double [4][4];
```

Следующий фрагмент кода инициализирует такое же количество памяти, но память под вторую размерность отводится вручную. Это сделано для того, чтобы наглядно показать, что матрица на самом деле представляет собой вложенные массивы.

```
double matrix [][] = new double [4][];
matrix [0] = new double[4];
matrix[1] = new double[4];
matrix[2] = new double[4], matrix[3] = { 0, 1, 2, 3 };
```

В следующем примере создается матрица размером 4 на 4 с элементами типа double, причем ее диагональные элементы (те, для которых x==y) заполняются единицами, а все остальные элементы остаются равными нулю.

```
class Matrix {
        public static void main(String args[]) { double m[][];
                m = new double[4][4];
                m[0][0] = 1;
                m[1][1] = 1;
                m[2][2] = 1;
```

```
                        m[3][3] = 1;
                        System.out.println(m[0][0] +" "+ m[0][1] +" "+
                                        m[0][2] +" "+ m[0][3]);
                        System.out.println(m[1][0] +" "+ m[1][1] +" "+
                                        m[1][2] +" "+ m[1][3]);
                        System.out.println(m[2][0] +" "+ m[2][1] +" "+
                                        m[2][2] +" "+ m[2][3]);
                        System.out.println(m[3][0] +" "+ m[3][1] +" "+
                                        m[3][2] +" "+ m[3][3]);
                        }
                }
```

Запустив эту программу, вы получите следующий результат:

```
        C : \> Java Matrix
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 0 1
```

Обратите внимание—если вы хотите, чтобы значение элемента было нулевым, вам не нужно его инициализировать, это делается автоматически.

Для задания начальных значений массивов существует специальная форма инициализатора, пригодная и в многомерном случае. В программе, приведенной ниже, создается матрица, каждый элемент которой содержит произведение номера строки на номер столбца. Обратите внимание на тот факт, что внутри инициализатора массива можно использовать не только литералы, но и выражения.

```
class AutoMatrix {
        public static void main(String args[]) { double m[][] = {
                { 0*0, 1*0, 2*0, 3*0 },
                { 0*1, 1*1, 2*1, 3*1 },
                { 0*2. 1*2, 2*2, 3*2 },
                { 0*3, 1*3. 2*3, 3*3 } }:
                System.out.println(m[0][0] +" "+m[0][1] +" "+
                                        m[0][2] +" "+ m[0][3]);
                System.out.println(m[1][0] +" "+m[1][1] +" "+
                                        m[1][2] +" "+ m[1][3]);
                System.out.println(m[2][0] +" "+m[2][1] +" "+
```

```
                                        m[2][2] +" "+ m[2][3]);
            System.out.println(m[3][0] +" "+m[3][1] +" "+
                                        m[3][2] +" "+ m[3][3]);
            }
    }
```

Запустив эту программу, вы получите следующий результат:

```
    C: \> Java AutoMatrix
    0 0 0 0
    0 1 2 3
    0 2 4 6
    0 3 6 9
```

8. Знай свои типы

Теперь вы знаете, как работать с восьмью простыми типами языка Java. Вы видели, как нужно создавать объекты этих типов и знаете разрядности каждого из них. Вы знаете, как эти типы взаимодействуют и какие из них подходят для арифметических вычислений. Вы познакомились с типом boolean и почувствовали, что от символов мало пользы пока нет возможности группировать их вместе, образуя слова—к этому вопросу мы вернемся в главе 9, где познакомимся со строками. Мы не обошли своим вниманием массивы и видели, как можно создавать массивы из массивов. В следующей главе мы научимся выполнять над всеми этими типами различные операции.