Лекция 2

Java переменные, операции

Javaпеременные

Java является строго типизированным языком. Это означает, что любая переменная илюбое выражение имеют известный тип еще на момент компиляции.

В языке Java все переменные должны быть объявлены, перед тем, как они будут использоваться.

Объявление переменных в java

Пример:

```
int x = 1;
int y = 2;
```

При объявлении переменной, в следующей последовательности указываются:

- **тип данных** (в данном примере -int- переменная содержит целое число),
- **имя переменной** (в данном примере имена - x и y),
- **начальное значение переменной** или, другими словами, **инициализация переменной**. В данном примере переменным x и y присвоены значения 1 и 2. Однако, это не является обязательным условием при объявлении переменной.

Пример: **объявление переменных без инициализации**:

```
int x;
int y;
```

После каждой строки при объявлении переменных необходимо ставить точку с запятой «;».

Если нужно **объявить несколько переменных одного типа**, то это также можно сделать одной строкой, указав имена переменных через запятую.

```
intx,y;
```

Правила именования переменных в java

1. Имя переменной должно **начинаться с буквы (маленькой) и состоять из букв (Unicode) цифр и символа подчеркивания «_»**. Технически возможно начать имя переменной также с «$» или «_», однако это запрещено соглашением по оформлению кода в Java (JavaCodeConventions). Кроме того, символ доллара «$», по соглашению, никогда не используется вообще. В соответствии с соглашением имя переменной должно начинаться именно с маленькой буквы (с заглавной буквы начинаются имена классов). Пробелы при именовании переменных не допускаются.

2. Имя переменной **не должно быть**ключевым или зарезервированным словом языка Java.

3. Имя переменной **чувствительно к регистру**. newVariable и newvariable- разные имена.

4. При выборе имени переменных, следует **использовать полные слова** вместо загадочных аббревиатур. Это сделает ваш код более удобным для чтения и понимания. Во многих случаях это также сделает ваш код самодокументируемым.

5. Если выбранное вами имя переменной состоит только из одного слова - запишите его маленькими буквами. Если оно состоит из более чем одного слова, то **отделяйте каждое последующее слово в имени переменной заглавной буквой.** Например: superCounter, myDomesticAnimal

6. Если переменная сохраняет постоянное значение, то каждое слово следует писать заглавными буквами и отделять при помощи символа подчеркивания. Пример:

```
static final intNUMBER_OF_HOURS_IN_A_DAY = 24
```

Типы данных в java

Каждая переменная и каждое выражение в Java обладает типом и этот тип строго определен.

Все типы данных разделяются на две группы: примитивные и ссылочные типы.

Примитивные типы данных

В Java существует 8 примитивных типов данных:

- `byte` (целые числа, 1 байт)
- `short` (целые числа, 2 байта)
- `int` (целые числа, 4 байта)
- `long` (целые числа, 8 байтов)
- `float` (вещественные числа, 4 байта)
- `double` (вещественные числа, 8 байтов)
- `char` (символ Unicode, 2 байта)
- `boolean` (значение истина/ложь, 1 байт)

Эти 8 типов служат основой для всех остальных типов данных. Примитивные типы обладают явным диапазоном допустимых значений.

byte— диапазон допустимых значений от -128 до 127

```
//объявление переменных типа byte.
bytegetByte, putByte;
// инициализация переменных
getByte = 0;
putByte = 0;
```

Переменные типа byte полезны при работе с потоком данных, который поступает из сети или файла.

short - диапазон допустимых значений от -32768 до 32767

```
//объявление и инициализация переменной типа short.
shortemployeeID = 0;
```

int— диапазон допустимых значений от -2147483648 до 2147483647

```
//объявление и инициализация переменной типа int.
intmax = 2147483647;
```

Тип intиспользуется чаще при работе с целочисленными данными, нежеlibyte и short, даже если их диапазона хватает. Это происходит потому, что при указании значений типabyte и shortв выражениях, их тип все равно автоматически повышается до int при вычислении.

long— диапазон допустимых значений от -9223372036854775808 до 9223372036854775807

```
//Использование переменных типа long.
longdays = getDays();
longseconds;
seconds = days * 24 * 60 * 60;
```

Тип удобен для работы с большими целыми числами.

float— диапазон допустимых значений от ~1,4*10^{-45} до ~3,4*10^{38}

```
//Объявление и инициализация переменных типа float.
floatusd = 31.24;
floateur = 44.03;
```

Удобен для использования, когда не требуется особой точности в дробной части числа.

double— диапазон допустимых значений от ~4,9*10^{-324} до ~1,8*10^{308}

```
//Объявление и инициализация переменных типа float.
doublepi = 3.14159;
```

Математические функции такие как sin(), cos(), sqrt() возвращают значение double.

char — символьный тип данных представляет собой один 16-битный Unicode символ. Он имеет минимальное значение ‘\ u0000′ (или 0), и максимальное значение ‘\ uffff’ (или 65535 включительно). Символы char можно задавать также при помощи соответствующих чисел. Например символ ‘Ы’ соответствует числу 1067. Рассмотримнапримере:

```
public static void main(String[] args) {
 char symb1=1067;
 char symb2 ='Ы';
 System.out.println("symb1 contains "+ symb1);
 System.out.println("symb2 contains "+ symb2);
}
```

Вывод этой программы будет:

```
symb1 contains Ы
symb2 contains Ы
```

Небольшой пример того, как узнать, какому числу соответствует символ. Основаннапретипованииданных.

```
public static void main(String[] args) {
 charch = 'J';
 intintCh = (int) ch;
 System.out.println("J corresponds with "+ intCh);
}
```

На выводе программа показывает, что символу 'J' соответствует число 74.

```
J correspondswith 74
```

boolean — предназначен для хранения логических значений. Переменные этого типа могут принимать только одно из 2х возможных значений true или false.

```
//Объявление и инициализация переменной типа boolean.
boolean b = true;
```

Тип String

Тип String не является примитивным типом данных, однако это один из наиболее используемых типов в Java. String предназначен для хранения строк текста. Несколько примеров использования String

```
//Создание строки с помощью конструктора
String myString = new String("The weather was fine");
//Можно также создать строку используя кавычки ""
String myString = "The weather was fine";
```

Для строк определен оператор «+»

```
public static void main(String[] args) {
 String s1 = "I have ";
 String s2 = " apples ";
 intnum = 3;
 String s = s1 + num + s2;
 System.out.println(s);
}
```

Выводпрограммы.

```
I have 3 apples
```

Ссылочные типы данных

В ссылочные типы входят все классы, интерфейсы, массивы. Описанный выше тип String также относится к ссылочным типам. Этот класс из стандартной библиотеки Java.

Также существуют классы-оболочки:

- Byte
- Short
- Integer
- Long
- Float
- Double
- Character
- Boolean

В отличие от примитивных типов, они пишутся с заглавной буквы. Эти типы соответствуют примитивным типам, однако являются ссылочными. Их классы содержат методы для преобразования типов, а также другие константы и методы полезные при работе с примитивными типами данных.

Операции языка Java

Большинство операций в Java аналогичны тем, которые применяются в других си-подобных языках. Есть унарные операции (выполняются над одним операндом), бинарные - над двумя операндами, а также тернарные - выполняются над тремя операндами. Операндом является переменная или значение (например, число), участвующее в операции. Рассмотрим все виды операций.

Арифметические операции

- +
  операция сложения двух чисел, например: *z=x+y*;
- -
  операция вычитания двух чисел: *z=x-y*;
- *
  операция умножения двух чисел: *z=x*y*;
- /
  операция деления двух чисел: *z=x/y*;
- %
  получение остатка от деления двух чисел: *z=x%y*;
- ++ (префиксный инкремент)

предполагает увеличение переменной на единицу, например, *z=++y* (вначале значение переменной *y* увеличивается на 1, а затем ее значение присваивается переменной *z*);

- ++ (постфиксный инкремент)

также, увеличение переменной на единицу, например, *z=y++* (вначале значение переменной *y* присваивается переменной *z*, а потом значение переменной *y* увеличивается на 1);

- -- (префиксный декремент)

уменьшение переменной на единицу, например, *z=--y* (вначале значение переменной y уменьшается на 1, а потом ее значение присваивается переменной *z*);

- -- (постфиксный декремент)

z=y-- (сначала значение переменной *y* присваивается переменной *z*, а затем значение переменной *y* уменьшается на 1).

Примеры:

```
int a1 = 3 + 4; // результат равен 7
int a2 = 10 - 7; //результат равен 3
int a3 = 10 * 5; //результат равен 50
double a4 = 14.0 / 5.0; //результат равен 2.8
double a5 = 14.0 % 5.0; //результат равен 4
int b1 = 5;
int c1 = ++b1; // c1=6; b1=6
int b2 = 5;
int c2 = b2++; // c2=5; b2=6
int b3 = 5;
int c3 = --b3; // c3=4; b3=4
```

```
int b4 = 5;
int c4 = b4--; // c4=5; b4=4
```

Операцию сложения также можно применять к строкам, в этом случае происходит конкатенация строк:

```
String hello = "hell to " + "world";
//результат равен "hell to world"
```

Логические операции над числами

Логические операции над числами представляют поразрядные операции. В данном случае числа рассматриваются в двоичном представлении, например, 2 в двоичной системе равно 10 и имеет два разряда, число 7 - 111 и имеет три разряда.

- & (логическое умножение)

Умножение производится поразрядно, и если у обоих операндов значения разрядов равно 1, то операция возвращает 1, иначе возвращается число 0. Например:

```
int a1 = 2; //010
int b1 = 5;//101
System.out.println(a1&b1); // результат 0

int a2 = 4; //100
int b2 = 5; //101
System.out.println(a2&b2); // результат 4
```

В первом случае у нас два числа 2 и 5. 2 в двоичном виде представляет число 010, а 5 - 101. Поразрядное умножение чисел (0·1, 1·0, 0·1) дает результат 000.

Во втором случае у нас вместо двойки число 4, у которого в первом разряде 1, так же как и у числа 5, поэтому здесь результатом операции (1·1, 0·0, 0·1) = 100 будет число 4 в десятичном формате.

- | (логическое сложение)

Данная операция также производится по двоичным разрядам, но теперь возвращается единица, если хотя бы у одного числа в данном разряде имеется единица (операция "логическое ИЛИ"). Например:

```
int a1 = 2; //010
int b1 = 5;//101
System.out.println(a1|b1); // результат 7 - 111
int a2 = 4; //100
int b2 = 5;//101
System.out.println(a2 | b2); // результат 5 - 101
```

- ^ (логическое исключающее ИЛИ)

Также эту операцию называют XOR, нередко ее применяют для простого шифрования:

```
intnumber=45;   //   1001   Значение,   которое   надо
зашифровать - в двоичной форме 101101
intkey=102; //Ключ шифрования - в двоичной системе
1100110
```

```
    intencrypt=number^key;   //Результатом   будет   число
1001011 или 75
    System.out.println("Зашифрованное число:"+encrypt);
    intdecrypt=encrypt^key; // Результатом будет исходное
число 45
    System.out.println("Расшифрованное число:"+decrypt)
```

Здесь также производятся поразрядные операции. Если у нас значения текущего разряда у обоих чисел разные, то возвращается 1, иначе возвращается 0. Например, результатом выражения 9^5 будет число 12. А чтобы расшифровать число, мы применяем обратную операцию к результату.

- ~ (логическое отрицание)

Поразрядная операция, инвертирующая все разряды числа: если значение разряда равно 1, то оно становится равным нулю, и наоборот.

```
    int a = 56;
    System.out.println(~a);
```

Операции сдвига

Операции сдвига также производятся над разрядами чисел. Сдвиг может происходить вправо и влево.

- a<<b - сдвигает число a влево на b разрядов. Например, выражение 4<<1 сдвигает число 4 (которое в двоичном представлении 100) на один разряд влево, в результате получается число 1000 или число 8 в десятичном представлении.
- a>>b - смещает число a вправо на b разрядов. Например, 16>>1 сдвигает число 16 (которое в двоичной системе 10000) на один разряд вправо, то есть в итоге получается 1000 или число 8 в десятичном представлении.
- a>>>b - в отличие от предыдущих типов сдвигов данная операция представляет беззнаковый сдвиг - сдвигает число a вправо на b разрядов. Например, выражение -8>>2 будет равно 1073741822.

Таким образом, если исходное число, которое надо сдвинуть в ту или другую строну, делится на два, то фактически получается умножение или деление на два. Поэтому подобную операцию можно использовать вместо непосредственного умножения или деления на два, так как операция сдвига на аппаратном уровне менее дорогостоящая операция в отличие от операции деления или умножения.

Операции сравнения

В операциях сравнения сравниваются два операнда, и возвращается значение типа boolean - true, если выражение верно, и false, если выражение неверно.

- ==

  данная операция сравнивает два операнда на равенство: *c=a==b*;

  c равно true, если *a* равно *b*, иначе *c* будет равно false;
- !=

  данная операция сравнивает два операнда на не равенство: *c=a!=b*;

  c равно true, если *a* не равно *b*, иначе *c* будет равно false;
- <

c=*a*<*b*; (*c* равно true, если *a* меньше *b*, иначе *c* будет равно false);

- >

c=*a*>*b*; (*c* равно true, если *a* больше *b*, иначе *c* будет равно false);

- <=

c=*a*<=*b*; (*c* равно true, если *a* меньше или равно *b*, иначе *c* будет равно false);

- >=

c=*a*>=*b*; (*c* равно true, если *a* больше или равно *b*, иначе *c* будет равно false).

Кроме собственно операций сравнения в Java также определены логические операторы, которые возвращают значение типа boolean. Выше мы рассматривали поразрядные операции над числами. Теперь же рассмотрим эти же операторы при операциях над булевыми значениями:

- |

c=*a*|*b*; (*c* равно true, если либо *a*, либо *b* (либо и *a*, и *b*) равны true, иначе *c* будет равно false);

- &

c=*a*&*b*; (*c* равно true, если и *a*, и *b* равны true, иначе *c* будет равно false);

- !

c!=*b*; (*c* равно true, если *b* равно false, иначе *c* будет равно false);

- ^

c=*a*^*b*; (*c* равно true, если либо *a*, либо *b* (но не одновременно) равны true, иначе *c* будет равно false);

- ||

c=*a*||*b*; (*c* равно true, если либо *a*, либо *b* (либо и *a*, и *b*) равны true, иначе *c* будет равно false);

- &&

c=*a*&&*b*; (*c* равно true, если и *a*, и *b* равны true, иначе *c* будет равно false).

Здесь у нас две пары операций | и || (а также & и &&) выполняют похожие действия, однако же они не равнозначны.

Выражение *c*=*a*|*b*; будет вычислять сначала оба значения - *a* и *b* и на их основе выводить результат.

В выражении же *c*=*a*||*b*; вначале будет вычисляться значение *a*, и если оно равно true, то вычисление значения *b* уже смысла не имеет, так как у нас в любом случае уже *c* будет равно true. Значение *b* будет вычисляться только в том случае, если *a* равно false.

То же самое касается пары операций &/&&. В выражении *c*=*a*&*b*; будут вычисляться оба значения - *a* и *b*.

В выражении же *c*=*a*&&*b*; сначала будет вычисляться значение *b*, и если оно равно false, то вычисление значения *b* уже не имеет смысла, так как значение *c* в любом случае равно false. Значение *b* будет вычисляться только в том случае, если *a* равно true.

Таким образом, операции || и && более удобны в вычислениях, позволяя сократить время на вычисление значения выражения и тем самым повышая

производительность. А операции | и & больше подходят для выполнения поразрядных операций над числами.

Примеры:

```
boolean a1=(5>6)||(4<6); //5>6 - false, 4<6 - true, поэтому возвращается true
boolean a2=(5>6)||(4>6); //5>6 - false, 4>6 - false, поэтому возвращается false
boolean a3=(5>6)&&(4<6); // 5>6 - false, 4<6 - true, поэтому возвращается false
boolean a4=(50>6)&&(4/2<3); //50>6 - true, 4/2<3 - true, поэтому возвращается true
boolean a5=(5>6)^(4<6); // 5>6 - false, 4<6 - true, поэтому возвращается true
boolean a6=(50>6)^(4/2<3); // 50>6 - true, 4/2<3 - true, поэтому возвращается false
```

Операции присваивания

В завершении рассмотрим операции присваивания, которые в основном представляют комбинацию простого присваивания с другими операциями:

- =
  просто приравнивает одно значение другому: *c*=*b*;
- +=
  c+=*b*; (переменной *c* присваивается результат сложения *c* и *b*);
- -=
  c-=*b*; (переменной *c* присваивается результат вычитания *b* из *c*);
- *=
  *c**=*b*; (переменной *c* присваивается результат произведения *c* и *b*);
- /=
  c/=*b*; (переменной *c* присваивается результат деления *c* на *b*);
- %=
  c%=*b*; (переменной *c* присваивается остаток от деления *c* на *b*);
- &=
  c&=*b*; (переменной *c* присваивается значение *c*&*b*);
- |=
  c|=*b*; (переменной *c* присваивается значение *c*|*b*);
- ^=
  c^=*b*; (переменной *c* присваивается значение *c*^*b*);
- <<=
  c<<=*b*; (переменной *z* присваивается значение *c*<<*b*);
- >>=
  c>>=*b*; (переменной *z* присваивается значение *c*>>*b*);
- >>>=
  c>>>=*b*; (переменной *c* присваивается значение *c*>>>*b*).

Условная операция

Это единственная тернарная операция, т.е. операция, имеющая три операнда. Соответственно, для нее используется не один знак операции, а два.

```
<условие> ? <выражение1> : < выражение2>
```

Если <условие> истинно, то результатом будет < выражение1>, иначе < выражение2>.

Например, "`a < b ? a : b`" вычисляет минимум из a и b.