Практическая работа №1

Примитивные типы данных

Одной из основных особенностей Java является то, что данный язык программирования является строго типизированным. Это значит, что каждая переменная и константа представляет определенный тип, и строго определен диапазон значений, которые она может хранить.

Систему встроенных базовых типов данных, которая используется для создания переменных в Java, представлена следующими типами:

- boolean – хранит значение true или false:

  ```
  boolean isActive = false;
  boolean isAlive = true;
  ```

- byte – хранит целое число от -128 до 127 и занимает 1 байт;

  ```
  byte a = 2;
  byte b = -7;
  ```

- short – хранит целое число от -32768 до 32767 и занимает 2 байта:

  ```
  short a = 3;
  short b = -8;
  ```

- int – хранит целое число от -2147483648 до 2147483647 и занимает 4 байта:

  ```
  int a = 4;
  int b = -9;
  ```

- long – хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт:

  ```
  long a = 5L;
  long b = -10L;
  ```

- float – хранит число с плавающей точкой от -3.4*1038 до 3.4*1038 и занимает 4 байта:

  ```
  float x = 8.5F;
  float y = -2.7F;
  ```

- double – хранит число с плавающей точкой от ±4.9*10-324 до ±1.8*10308 и занимает 8 байт:

  ```
  double x = 8.5;
  double y = -2.7;
  ```

- char – хранит одиночный символ в кодировке UTF-16 и занимает 2 байта, поэтому диапазон хранимых в нём значений – целые числа от 0 до 65535.

Все целочисленные литералы, например, числа 10, 4, -5, воспринимаются как значения типа int, однако мы можем присваивать целочисленные литералы другим

целочисленным типам: byte, long, short. В этом случае Java автоматически осуществляет соответствующие преобразования:

```
byte a = 1;
short b = 2;
long c = 2121;
```

Однако если мы захотим присвоить переменной типа long число, которое выходит за пределы допустимых значений для типа int, то мы столкнемся с ошибкой во время компиляции:

```
long num = 2147483649;
```

Здесь число 2147483649 является допустимым для типа long, но выходит за предельные значения для типа int. И, так как все целочисленные значения по умолчанию расцениваются как значения типа int, компилятор укажет нам на ошибку. Чтобы решить проблему, надо добавить к числу суффикс l или L, который указывает, что число представляет тип long:

```
long num = 2147483649L;
```

Как правило, значения для целочисленных переменных задаются в десятичной системе счисления, однако мы можем применять и другие системы счисления. Например:

```
int num13 = 0b1101; // 2-я система, число 13
int num8 = 010; // 8-я система, число 8
int num111 = 0x6F; // 16-я система, число 111
```

Для задания восьмеричного значения после символа 0 указывается число в восьмеричном формате. Таким же образом двоичное значение указывается после символов 0b, а шестнадцатеричное значение - после символов 0x.

Также целые числа поддерживают разделение разрядов числа с помощью знака подчеркивания:

```
int x = 123_456;
int y = 234_567__789;
System.out.println(x);  // 123456
System.out.println(y);  // 234567789
```

При присвоении переменной типа float дробного литерала с плавающей точкой, например, 3.1, 4.5 и т.д., Java автоматически рассматривает этот литерал как значение типа double. Чтобы указать, что данное значение должно рассматриваться как значение типа float, нам надо использовать суффикс f:

```
float fl = 30.6f;
double db = 30.6;
```

И хотя в данном случае обе переменных имеют практически одно и то же значение, эти значения будут по-разному рассматриваться и будут занимать разное место в памяти.

В качестве значения переменная символьного типа получает одиночный символ, заключенный в одинарные кавычки:

```
char ch='e';
```

Кроме того, переменной символьного типа также можно присвоить целочисленное значение от 0 до 65535. В этом случае переменная опять же будет хранить символ, а целочисленное значение будет указывать на номер символа в таблице символов Unicode (UTF-16). Например:

```
char ch=102; // символ 'f'
System.out.println(ch);
```

Еще одной формой задания символьных переменных является шестнадцатеричная форма: переменная получает значение в шестнадцатеричной форме, которое следует после символов "\u". Например, данная переменная тоже будет хранить символ 'f':

```
char ch='\u0066'; // символ 'f'
System.out.println(ch);
```

Символьные переменные не стоит путать со строковыми, значение 'a' не идентично значению "a". Строковые переменные представляют объект String, который в отличие от char не является примитивным типом в Java:

```
String hello = "Hello!";
System.out.println(hello);
```

Кроме отображаемых символов, которые представляют буквы, цифры, знаки препинания и прочие символы, есть специальные символы, которые называют управляющими последовательностями (escape-последовательностями).

Управляющие последовательности, используемые в языке Java:

Последовательность	**Описание**	**Код символа**
\b	возврат на одну позицию	\u0008
\t	табуляция	\u0009
\n	перевод строки	\u 000A
\f	прогон страницы	\u000C
\r	возврат каретки	\u000D
\”	двойная кавычка	\u0022
\’	одинарная кавычка	\u0027
\\	обратная косая черта	\u005C

Например, так используется самая популярная последовательность - "\n", выполняющая перевод курсора на следующую строку:

```
String text = "Hello,\nworld!";
System.out.println(text);
```

Результат выполнения данного кода:

```
Hello,
world!
```

Начиная с версии 15, Java поддерживает тестовые блоки – многострочный текст, заключенный в тройные кавычки. Например, выведем большой многострочный текст:

```
String text = "В век высоких технологий\n" +
        "Без программ не обойтись,\n" +
        "Программисты ежедневно\n" +
        "Легче делают нам жизнь!""";
System.out.println(text);
```

С помощью операции + мы можем присоединить к одному тексту другой, причем продолжение текста может располагаться на следующей строке. Чтобы при выводе текста происходил перенос на следующую строку, применяется последовательность \n.

```
В век высоких технологий
Без программ не обойтись,
Программисты ежедневно
Легче делают нам жизнь!
```

Текстовые блоки, которые появились в JDK15, позволяют упростить написание многострочного текста:

```
String text = """
        В век высоких технологий
        Без программ не обойтись,
        Программисты ежедневно
        Легче делают нам жизнь!
        """;
System.out.println(text);
```

Весь текстовый блок оборачивается в тройные кавычки, при этом не надо использовать соединение строк или последовательность \n для их переноса. Результат выполнения программы будет тем же, что и в примере выше.

Выражения и операции

Выражения являются основными составляющими любого Java-приложения. Выражения строятся с использованием значений, переменных, операторов и вызовов методов. *Операторы* в языке Java – это специальные символы, которые сообщают о том, что необходимо выполнить какую-либо операцию с некоторыми данными – операндами. Операндом является переменная или значение, участвующее в операции. Различают *унарные* операции – они выполняются над одним операндом, *бинарные* - над двумя операндами, а также *тернарные* - выполняются над тремя операндами. Операторы, которые расположены перед операндами, называются *префиксными*, операторы, которые расположены после операндов – *постфиксными*, операторы, расположенные между двумя операндами - *инфиксными* операторами.

Операторы языка Java можно разделить на следующие категории: арифметические операторы, логические операторы, операторы сравнения, побитовые операторы и операторы присваивания, а также тернарный оператор.

Арифметические операции служат для выполнения арифметических действий над числами:

- операция сложения двух чисел:
```
int a = 10;
int b = 7;
int c = a + b;  // 17
int d = 4 + b;  // 11
```
- операция вычитания двух чисел:
```
int a = 10;
int b = 7;
int c = a - b;  // 3
int d = 4 - a;  // -6
```
- операция умножения двух чисел:
```
int a = 10;
int b = 7;
int c = a * b;  // 70
int d = b * 5;  // 35
```
- операция деления двух чисел:
```
int a = 20;
int b = 5;
int c = a / b;  // 4
double d = 22.5 / 4.5;  // 5.0
```
- получение остатка от деления двух чисел:
```
int a = 33;
int b = 5;
int c = a % b;  // 3
int d = 22 % 4; // 2 (22 - 4*5 = 2)
```

При делении стоит учитывать, что если в операции участвуют два целых числа, то результат деления будет округляться до целого числа, даже если результат присваивается переменной float или double:

```
double k = 10 / 4;     // 2
```

Чтобы результат представлял число с плавающей точкой, один из операндов также должен представлять число с плавающей точкой:

```
double k = 10.0 / 4;     // 2.5
```

Также есть две унарные арифметические операции, которые производятся над одним числом: ++ (инкремент) и -- (декремент). Каждая из операций имеет две разновидности: префиксная и постфиксная:

- префиксный инкремент – предполагает увеличение переменной на единицу (сначала значение переменной a увеличивается на 1, а затем ее значение присваивается переменной b):

```
int a = 8;
int b = ++a;
System.out.println(a);  // 9
System.out.println(b);  // 9
```

- постфиксный инкремент – также предполагает увеличение переменной на единицу (сначала значение переменной a присваивается переменной b, а потом значение переменной a увеличивается на 1):

```
int a = 8;
int b = a++;
System.out.println(a);  // 9
System.out.println(b);  // 8
```

- префиксный декремент – предполагает уменьшение переменной на единицу (сначала значение переменной a уменьшается на 1, а потом ее значение присваивается переменной b):

```
int a = 8;
int b = --a;
System.out.println(a);  // 7
System.out.println(b);  // 7.
```

- постфиксный декремент – также предполагает уменьшение переменной на единицу (сначала значение переменной a присваивается переменной b, а затем значение переменной a уменьшается на 1):

```
int a = 8;
int b = a--;
System.out.println(a);  // 7
System.out.println(b);  // 8
```

.

Одни операции имеют больший приоритет, чем другие, и поэтому выполняются раньше. Операции в порядке уменьшения приоритета:

1. ++ (инкремент), -- (декремент)
2. * (умножение), / (деление), % (остаток от деления)
3. + (сложение), - (вычитание)

Приоритет операций следует учитывать при выполнении набора арифметических выражений:

```
int a = 8;
int b = 7;
int c = a + 5 * ++b;
System.out.println(c);  // 48
```

Сначала будет выполняться операция инкремента ++b, которая имеет больший приоритет - она увеличит значение переменной b и возвратит его в качестве результата. Затем выполняется умножение 5 * ++b, и только в последнюю очередь выполняется сложение a + 5 * ++b.

Скобки позволяют переопределить порядок вычислений:

```
int a = 8;
int b = 7;
int c = (a + 5) * ++b;
System.out.println(c);  // 104
```

Несмотря на то, что операция сложения имеет меньший приоритет, сначала будет выполняться именно сложение, а не умножение, так как операция сложения заключена в скобки.

Помимо приоритета, операции отличаются таким понятием как ассоциативность. Когда операции имеют один и тот же приоритет, порядок вычисления определяется ассоциативностью операторов. В зависимости от ассоциативности есть два типа операторов:

- левоассоциативные операторы, которые выполняются слева направо;
- правоассоциативные операторы, которые выполняются справа налево.

Все арифметические операторы, кроме префиксного инкремента и декремента, являются левоассоциативными, то есть выполняются слева направо.

Следует отметить, что числа с плавающей точкой не подходят для финансовых и других вычислений, где ошибки при округлении могут быть критичными. Например:

```
double d = 2.0 - 1.1;
System.out.println(d);  //0.8999999999999999
```

Подобные ошибки точности возникают из-за того, что на низком уровне для представления чисел с плавающей точкой применяется двоичная система, однако для числа 0.1 не существует двоичного представления, также как и для других дробных значений. Поэтому в таких случаях обычно применяется класс BigDecimal, который позволяет обойти подобные ситуации.

Поразрядные операции выполняются над отдельными разрядами или битами чисел. В данных операциях в качестве операндов могут выступать только целые числа.

Каждое число имеет определенное двоичное представление. Например, число 4 в двоичной системе представлено значением 100, число 5 – 101, и так далее.

К примеру, возьмем следующие переменные:

```
byte b = 7;     // 0000 0111
short s = 7;    // 0000 0000 0000 0111
```

Тип byte занимает 1 байт или 8 бит, соответственно представлен 8 разрядами. Поэтому значение переменной b в двоичном коде будет равно 00000111. Тип short занимает в памяти 2 байта или 16 бит, поэтому число данного типа будет представлено 16 разрядами. И в данном случае переменная s в двоичной системе будет иметь значение 0000 0000 0000 0111.

Для записи чисел со знаком в Java применяется дополнительный код, при котором старший разряд является знаковым. Если его значение равно 0, то число положительное, и его двоичное представление не отличается от представления беззнакового числа. Например, 0000 0001 в десятичной системе имеет значение 1.

Если старший разряд равен 1, то мы имеем дело с отрицательным числом. Например, 1111 1111 в десятичной системе представляет -1. Соответственно, 1111 0011 представляет -13.

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 равно 111 и имеет три разряда.

- логическое отрицание – поразрядная операция, которая инвертирует все разряды числа: если значение разряда равно 1, то оно становится равным нулю, и наоборот:

  ```
  byte a = 12;                  // 0000 1100
  System.out.println(~a);  //  1111 0011   или -13
  ```

- логическое умножение – производится поразрядно, и если у обоих операндов значения разрядов равно 1, то операция возвращает 1, иначе возвращается число 0.

  ```
  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
  ```

- логическое сложение – также производится по двоичным разрядам, но теперь возвращается 1, если хотя бы у одного числа в данном разряде имеется 1.

```
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 – если у нас значения текущего разряда у обоих чисел разные, то возвращается 1, иначе возвращается 0. Также эту операцию называют XOR, нередко ее применяют для простого шифрования:

```
int number = 45; // 1001 Значение, которое надо зашифровать - 101101
int key = 102; //Ключ шифрования - 1100110
int encrypt = number ^ key; //Результатом будет число 1001011 или 75
System.out.println("Зашифрованное число: " +encrypt);
int decrypt = encrypt ^ key; // Результатом будет исходное число 45
System.out.println("Расшифрованное число: " + decrypt);
```

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, в результате 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 сторону, 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 ресурсоёмкая операция, чем операции деления или умножения.

Условные выражения представляют собой некоторое условие и возвращают значение типа boolean, то есть значение true (если условие истинно), или значение false (если условие ложно). К условным выражениям относятся операции сравнения и логические операции.

Операции сравнения сравнивают два операнда и возвращают значение типа boolean: true, если выражение верно, и false, если выражение неверно.

- равно – сравнивает два операнда на равенство и возвращает true, если операнды равны, и false, если операнды не равны:

```
int a = 10;
int b = 4;
```

```
boolean c = a == b;        // false
boolean d = a == 10;       // true
```

- не равно - сравнивает два операнда и возвращает true, если операнды не равны, и false, если операнды равны:

```
int a = 10;
int b = 4;
boolean c = a != b;        // true
boolean d = a != 10;       // false
```

- меньше – возвращает true, если первый операнд меньше второго, иначе возвращает false:

```
int a = 10;
int b = 4;
boolean c = a < b;   // false
```

- больше – возвращает true, если первый операнд больше второго, иначе возвращает false:

```
int a = 10;
int b = 4;
boolean c = a > b;   // true
```

- меньше или равно - возвращает true, если первый операнд меньше второго или равен второму, иначе возвращает false:

```
boolean c = 10 <= 10;    // true
boolean b = 10 <= 4;     // false
boolean d = 10 <= 20;    // true
```

- больше или равно – возвращает true, если первый операнд больше второго или равен второму, иначе возвращает false:

```
boolean c = 10 >= 10;    // true
boolean b = 10 >= 4;     // true
boolean d = 10 >= 20;    // false
```

Логические операции также представляют условие, возвращают значение типа boolean и обычно объединяют несколько операций сравнения. К логическим операциям относят следующие:

- 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 = 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 = a&&b - c равно true, если и a, и b равны true, иначе c будет равно false.

Пара операций | и ||, а также & и && выполняют похожие действия, однако они не равнозначны. В сокращенных операциях || и && правый операнд вычисляется только в

том случае, если от него зависит результат операции – если левый операнд операции || имеет значение false, или левый операнд операции && имеет значение true:

```
boolean a1 = (5 > 6) || (4 < 6); // 5 > 6 - false, 4 < 6 - true, значение выражения - true
boolean a2 = (5 > 6) || (4 > 6); // 5 > 6 - false, 4 > 6 - false, значение выражения - false
boolean a3 = (50 > 6) && (4 / 2 < 3); // 50 > 6 - true, 4/2 < 3 - true, значение выражения - true
boolean a4 = (5 > 6) && (4 < 6); // 5 > 6 – false,  4 < 6 – значение не вычисляется, значение выражения – false
boolean a5 = (5 > 6) ^ (4 < 6); // 5 > 6 - false, 4 < 6 – true, результат - true
boolean a6 = (50 > 6) ^ (4 / 2 < 3); // 50 > 6 - true, 4/2 < 3 - true, результат - false
```

В завершение рассмотрим составные операции присваивания, которые представляют комбинацию простого присваивания с другими операциями:

- 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 c присваивается значение c<<b;
- c >>= b – переменной c присваивается значение c>>b;
- c >>>= b – переменной c присваивается значение c>>>b.

Практические задания

1. Записать выражение, позволяющее вычислить:
   а) значение функции $x = 12a^2 + 7a - 16$ при известном значении a;
   б) значение функции $y = 7x^2 - 3x + 6$ при известном значении x.
2. Записать выражение, позволяющее вычислить для четырёхугольника со сторонами a и b:
   а) периметр;
   б) площадь.
3. Известно значение температуры по шкале Цельсия. Записать выражение, позволяющее вычислить соответствующее значение температуры по шкале:
   а) Фаренгейта - исходное значение температуры необходимо умножить на 1,8 и к результату прибавить 32;
   б) Кельвина - абсолютное значение нуля соответствует 273,15 градуса по шкале Цельсия.