Лекция 5

Арифметические и логические операции. Массивы.

Арифметические операции позволяют производить весь основной набор математических действий над числами. В свою очередь все арифметические операции можно разделить еще на 2 типа - простые и составные, основным отличием которых является тот момент, что составные операции производят дополнительное действие - присваивание.

Так же стоит отметить, что операции сложения (+) и вычитания (-) так же могут быть использованы как унарные операции - операции, которые работают со знаком числа.

Основные типы арифметических операций представлены в таблице ниже.

Арифметические операции

Операция	Описание
+	Сложение (унарный плюс)
-	Вычитание (унарный минус)
*	Умножение
/	Деление
%	Деление по модулю
++	Инкремент
--	Декремент
+=	Сложение с присваиванием
-=	Вычитание с присваиванием
*=	Умножение с присваиванием
/=	Деление с присваиванием
%=	Деление по модулю с присваиванием

Простые арифметические операции, как уже было сказано выше, производят основные математические операции над числовыми типами данных, составные же имеют дополнительную операцию - присваивание.

Стоит отметить, что составные операции имеют дополнительное преимущество - кроме укороченной записи они так же имеют преимущества по скорости работы, над простыми операциями, например запись вида a+=10; будет выполнена несколько быстрее, нежели набор операций, аналогичный по действиям (a = a+10;).

Операции инкремента (приращения на 1) и декремента так же упрощают запись кода, а так же представлены двумя формами - постфиксной и префиксной.

Префиксная форма записи подразумевает, что оператор инкремента/декремента стоит перед операндом, над которым производится действие, тогда как постфиксная предполагает, что оператор стоит после.

Несмотря на то, что префиксная и постфиксная запись выполняют одно и то же действие над числом, порядок операция в выражениях с префиксной и постфиксной формой записи будут разными. Так запись вида d = ++a; будет интерпретирована следующим образом: сначала будет произведен инкремент, а только затем присвоение, тогда как в аналогичной записи, то только с постфиксным вариантом инкремента будет произведено сначала присвоение, а только затем инкремент. Поэтому при построении выражений всегда стоит учитывать особенности той или иной формы записи.

Битовые операции.

Битовые операции - это операции, которые производятся на уровне битовых представлений целых чисел (отдельных битах числа). Среди битовых операций можно так же выделить 2 типа - логические и сдвиговые.

Логические производят ряд операций над битами числа, согласно правилам булевой алгебры, тогда как сдвиговые манипулируют содержимым, производя сдвиг битов в определенной последовательности.

Далее в таблице рассмотрим основные битовые операции.

Битовые операции

Операция	Описание
~	Бобитовое НЕ (унарная операция)
&	Побитовое И
|	Побитовое ИЛИ
^	Исключающее ИЛИ
>>	Битовый сдвиг вправо
>>>	Беззнаковый сдвиг вправо
<<	Сдвиг влево
&=	И с присваиванием
|=	ИЛИ с присваиванием
^=	Исключающее или с присваиванием
>>=	Сдвиг вправо с присваиванием
>>>=	Беззнаковый сдвиг вправо с присваиванием
<<=	Сдвиг влево с присваиванием

Логические битовые операции производятся над целочисленными значениями и могут быть записаны в качестве выражения как с несколькими операндами, так и в унарной форме (применимо к операции НЕ).

Поразрядная унарная операция НЕ. Эта операция обозначается знаком ~ и называется также поразрядным отрицанием, инвертируя все двоичные разряды своего операнда. Например, число 42, представленное в следующей двоичной форме:

00101010

преобразуется в результате выполнения поразрядной унарной операция НЕ в сле-дующую форму:

1101 0101

Поразрядная логическая операция И.

При выполнении поразрядной логической операции И, обозначаемой знаком &, в двоичном разряде результата устанавливается 1 лишь в том случае, если соответствующие двоичные разряды в операндах также равны 1. Во всех остальных случаях в двоичном разряде результата устанавливается 0, как показано ниже.

00101010 **42**

& 0000 1111 **15**

---

00 001010 **10**

Поразрядная логическая операция ИЛИ.

При выполнении поразрядной логической операции ИЛИ, обозначаемой зна-ком |, в двоичном разряде результата устанавливается 1, если соответствующий двоичный разряд в любом из операндов равен 1, как показано ниже.

```
  00101010    42
| 00001111    15
-----------------
  00101111    47
```

Поразрядная логическая операция исключающее ИЛИ.

При выполнении поразрядной логической операции исключающее ИЛИ, обозна-чаемой знаком ^, в двоичном разряде результата устанавливается 1, если двоичный разряд только в одном из операндов равен 1, а иначе в двоичном разряде результата устанавливается 0, как показано ниже. Приведенный ниже пример демонстрирует также полезную особенность поразрядной логической исключающее операции ИЛИ. Обратите внимание на инвертирование последовательности двоичных разрядов чис-ла 42 во всех случаях, когда в двоичном разряде второго операнда установлена 1. А во всех случаях, когда в двоичном разряде второго операнда установлен 0, двоичный раз- ряд первого операнда остается без изменения. Этим свойством удобно пользоваться при манипулировании отдельными битами числовых значений.

```
  00101010    42
| 00001111    15
----------------------
  00100101    37
```

Сдвиговые операции. Сдвиг влево.

Все биты смещаются влево. Число справа дополняется нулем. Операция используется для быстрого умножения на 2. Если оператор применяется к числу, умножение на 2 которого будет больше максимального значения int (2147483647), то в результате будет отрицательное число. Это происходит потому, что краний левый бит, который отвечает за знак числа, выставляется в единицу, что соответствует отрицательным числам.

Сдвиг вправо

Все биты смещаются вправо. Число слева дополняется нулем, если число положительное и единицей, если отрицательное. Операция используется для быстрого деления на 2. Если делится нечетное число, то остаток отбрасывается для положительных чисел и сохраняется для отрицательных.

Беззнаковый сдвиг вправо.

Все биты смещаются вправо, число слева дополняется нулем, даже если операция выполняется с отрицательными числами. Отсюда и название оператора — беззнаковый. В результате применения оператора всегда получается положительное число, т.к. в Java левый бит отвечает за знак числа. Операция так же, как и знаковый оператор сдвига вправо, соответствует делению числа на два за исключением первого сдвига в отрицательном числе.

Операции отношения.

Операции отношения, называемые иначе операциями сравнения, определяют отно-шение одного операнда к другому. В частности, они определяют равенство и упо-рядочение. Результатом выполнения этих операций оказывается логическое значение. Чаще всего операции отношения применяются в выражениях, управляющих ус-ловным оператором if и различными операторами цикла.

Правила сравнения полностью соответствуют математическим для простых типов данных.

Таблица операторов отношения представлена ниже.

Операции отношения

Операция	Описание
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше либо равно
<=	Меньше либо равно

Массивы.

Массив - это именованное множество элементов (переменных) одинакового типа. Каждая переменная в массиве называется элементом массива. Данная структура поддерживает индексацию и произвольный доступ к элементам, это значит, что имея порядковый номер элемента в массиве (индекс) мы можем как получить его значение, так и модифицировать его.

Индексация массивов начинается с 0, это значит, что первый элемент массива будет иметь индекс 0, тогда как последний будет равен количеству элементов -1.

Для получения количества элементов в массиве используется поле length, которое содержит наш массив.

Объявление массива в коде возможно несколькими способами:

int[] arr; - где int - тип данных элементов, которые содержатся в массиве, [] - квадратные скобки, как признак того, что это массив, arr - имя массива, которое будет использовано в коде.

int [] arr = new int[n]; - форма записи, указывающая на то, что у нас создается новый масиив на n элементов, оператор new используется для выделения памяти под новую переменную.

int [] arr = {0, 1, 2, …… n}; - форма записи, предусматривающая инициализацию массива, длина массива будет равна количеству элементов, указанных в фигурных скобках.

В Java выход индекса за границы массива строго проверяется, в результате подобной операции возникает исключение - **ArrayOutOfBoundsException.**

Многомерные массивы это массивы, которые по сути состоят из массивов. Примером можем рассмотреть двумерные массивы.

Инициализация многомерных массивов идентична инициализации одномерных массивов, за тем исключением, что для каждого элемента массива задается массив значений.

Форма объявления массива может выглядеть следующим образом:

int [][] array = new int [m][n]; где m и n сообветственно являются количеством элементов в каждом массиве.

На данном примере видно, что элементы из множества m являются массивами, которые в свою очередь содержат n элементов.

Управляющие конструкции. Циклы.

Цикл - это некая последовательность однотипных операций, которые следуют друг за другом. Для реализации данных действий в Java имеются 3 основные конструкции :

for(переменная; условие существования; изменение переменной)

while(условие существования)

do while(условие существования).

Теперь разберем каждую конструкцию подробнее.

Цикл for.

Конструкция for имеет следующую форму записи:

for(переменная; условие существования; изменение переменной)

где переменная - переменная, которая будет использоваться для перехода по шагам (итерациям) цикла, условие существования - условие, которое определяет, до каких пор будет работать цикл, изменение переменной - вариант изменения переменной, по которому будет производиться переход от итерации к итерации.

Форма записи цикла выглядит следующим образом:

```
for(int i = 0; i < 10; i++){

[код, выполняемый в цикле]

}
```

где фигурные скобки ограничивают набор действий, которые будут выполняться в цикле (код, выполняемый в цикле). Если участок кода, который должен выполняться в цикле содержит в себе единичную операцию, то фигурные скобки можно опустить, т.е. в таком случае будет в цикле будет выполнена первая операция, идущая за конструкцией for.

Цикл while.

Конструкция while выполняет циклическое действие до тех пор, пока истинно условие выполнения.

Форма записи выглядит следующим образом:

while(условие){

[код, выполняющийся в цикле]

}

где условие - некое логическое выражение, результатом которого является значение true или false. Цикл выполняется до тех пор, пока выражение истинно.

Цикл do while.

Конструкция do while выполняет те же действия, но при этом гарантирует, что цикл будет иметь хотя бы одну итерацию, так как первая проверка истинности выражения производится после выполнения первой итерации цикла.

Форма записи выглядит следующим образом:

do{

[код, выполняющийся в цикле]

}while(условие);

где условие - некое логическое выражение, результатом которого является значение true или false. Цикл выполняется до тех пор, пока выражение истинно.