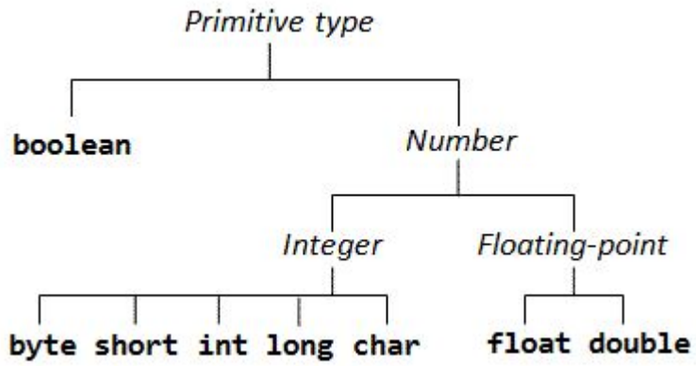
II. Типы, переменные, управляющие инструкции

2. Примитивные типы



	Примитивный тип предопределён в языке и для него зарезервировано ключевое слово. Всего существует 8 примитивных типов: булевский примитивный тип boolean и 7 численных типов. Численные типы делятся на целочисленные типы byte, short, char, int, long и вещественные типы float и double.

	Примитивы могут быть операндами операторов. Существует 1 оператор присваивания, 12 арифметических операторов, 12 логических операторов, 6 операторов сравнения и 13 битовых операторов.

Примитивные типы

Тип	Размер	Мин. значение	Макс. значение	Значение по умолчанию
byte	8 бит	-128	127	0
short	16 бит	-32,768	32,767	0
char	16 бит	'\u0000' (0)	'\uffff' (65535)	'\u0000'(0)
int	32 бита	-2,147,483,648	2,147,483,647	0
long	64 бита	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	0L
float	32 бита	IEEE754	IEEE754	0.0f
double	64 бита	IEEE754	IEEE754	0.0d
boolean	-	-	-	false

Диапазоны значений и размер

```
public class RangeSizeDemo {

   public static void main(String[] args) {

      System.out.println("byte min: " + Byte.MIN_VALUE);
      System.out.println("byte max: " + Byte.MAX_VALUE);
      System.out.println("byte length: " + Byte.SIZE);

      System.out.println("short min: " + Short.MIN_VALUE);
      System.out.println("short max: " + Short.MAX_VALUE);
      System.out.println("short length: " + Short.SIZE);

      System.out.println("char min: " + (int) Character.MIN_VALUE);
      System.out.println("char max: " + (int) Character.MAX_VALUE);
      System.out.println("char length: " + Character.SIZE);

      System.out.println("int min: " + Integer.MIN_VALUE);
      System.out.println("int max: " + Integer.MAX_VALUE);
      System.out.println("int length: " + Integer.SIZE);

      System.out.println("long min: " + Long.MIN_VALUE);
      System.out.println("long max: " + Long.MAX_VALUE);
      System.out.println("long length: " + Long.SIZE);

      System.out.println("float min approx: " + Float.MIN_VALUE);
      System.out.println("float max approx: " + Float.MAX_VALUE);
      System.out.println("float length: " + Float.SIZE);

      System.out.println("double min approx: " + Double.MIN_VALUE);
      System.out.println("double max approx: " + Double.MAX_VALUE);
      System.out.println("double length: " + Double.SIZE);
   }
}
```

Диапазоны значений и размер

```
byte min: -128
byte max: 127
byte length: 8
short min: -32768
short max: 32767
short length: 16
char min: 0
char max: 65535
char length: 16
int min: -2147483648
int max: 2147483647
int length: 32
long min: -9223372036854775808
long max: 9223372036854775807
long length: 64
float min approx: 1.4E-45
float max approx: 3.4028235E38
float length: 32
double min approx: 4.9E-324
double max approx: 1.7976931348623157E308
double length: 64
```

Примитивные переменные

```
[byte|short|char|int|long|float|double|boolean] variable [ = literal | = expression ];
```

Значения переменных примитивных типов хранятся по значению. Переменная примитивного типа всегда содержит значение именно этого типа. При объявлении переменная примитивного типа может быть сразу же инициализирована. Переменные примитивных типов могут быть инициализированны примитивными литералами. Примитивные литералы это значения примитивных типов в исходном коде программы. Для примитивов существуют четыре вида литералов: целочисленные литералы, литералы с плавающей точкой, символьные литералы и булевские литералы.

Примитивы и литералы

```
public class LiteralDemo {

   public static void main(String[] args) {

      String name = "Harry Hacker";
      char gender = 'm';
      boolean isMarried = true;
      byte numChildren = 2;
      short yearOfBirth = 1987;
      int salary = 30000;
      long netAsset = 8234567890L;
      double weight = 88.88;
      float gpa = 4.58f;

      System.out.println("Name: " + name);
      System.out.println("Gender: " + gender);
      System.out.println("Is married: " + isMarried);
      System.out.println("Number of children: " + numChildren);
      System.out.println("Year of birth: " + yearOfBirth);
      System.out.println("Salary: " + salary);
      System.out.println("Net Asset: " + netAsset);
      System.out.println("Weight: " + weight);
      System.out.println("GPA: " + gpa);
   }
}
```

```
Name: Harry Hacker
Gender: m
Is married: true
Number of children: 2
Year of birth: 1987
Salary: 30000
Net Asset: 8234567890
Weight: 88.88
GPA: 4.58
```

Булевский тип

Булевские литералы

Тип	Литерал	Пример
boolean	true	true
boolean	false	false

Булевские литералы имеют тип boolean и могут использоваться только для инициализации переменных типа boolean. Есть всего два булевских литерала true и false.

Тип boolean не имеет никакого отношения к целочисленным типам. Переменную типа boolean нельзя инициализировать 0 или 1.

Булевы логические операторы

Оператор	Результат
&	Логическое и
|	Логическое или
^	Логическое исключающее или (XOR)
||	Логическое или быстрой оценки
&&	Логическое и быстрой оценки
!	Отрицание
&=	Логическое и с присваиванием
|=	Логическое или с присваиванием
^=	Логическое ислючающее или с присваиванием
==	Равно
!=	Не равно
?:	Тернарный оператор if-then-else

У тернарного оператора первый операнд должен быть boolean другие операнды и результат могут быть любого типа. У других операторов операндами могут быть только boolean. Результат тоже boolean.

Таблица истинности

```
public class TruthTableDemo {

   public static void main(String[] args) {

      System.out.println("L\tR\tAND\tOR\tXOR\tNOT");
      printLine(true, true);
      printLine(true, false);
      printLine(false, true);
      printLine(false, false);
   }

   static void printLine(boolean l, boolean r) {

      System.out.println(l + "\t" + r + "\t" + (l & r) + "\t" + (l | r) + "\t" + (l ^ r));
   }
}
```

```
L       R       AND     OR      XOR
true    true    true    true    false
true    false   false   true    true
false   true    false   true    true
false   false   false   false   false
```

Целочисленные типы

Целочисленные литералы

Тип	Суффикс	Пример
int		-123412
long	L или l	12341234123L

Целочисленные литералы без суффикса (по умолчанию) имеют тип int и могут использоваться для инициализации переменных типа byte, short, char, int и long при условии что значение литерала находится в диапазоне значений типа. Целочисленные литералы с суффиксом L или l имеют тип long и могут быть использованы для инициализации переменных типа long. Значение литерала должно быть в соответствующем диапазоне иначе произойдёт ошибка компиляции.

Суффикс l использовать не следует. Его легко перепутать с цифрой 1. Вместо него следует использовать суффикс L.

Система счисления	Префикс	Пример
десятичная		27
двоичная	0b	0b11011
восьмиричная	0	033
шестнадцатиричная	0x	0x1b

Необходимо помнить что целочисленный литерал начинающийся с 0 это число в восьмиричной системе.

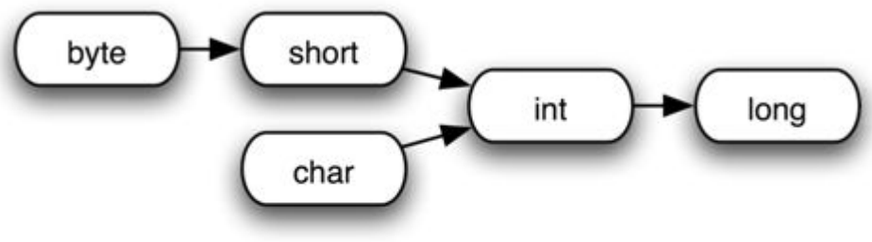
Символьные литералы

Тип	Литерал	Описание
char	'\uxxxx'	Символ с кодом xxxx в кодировке Unicode
char	'X'	Символ X

Символьные литералы имеют тип char и могут использоваться для инициализации целочисленных переменных при условии что значение литерала находится в диапазоне значений типа.

Литерал	Значение	Эквивалент
'\b'	возврат каретки	'\u0008'
'\t'	табулятор	'\u0009'
'\n'	переход строки	'\u000A'
'\f'	подача бланка	'\u000C'
'\r'	возврат каретки	'\u000D'
'\"'	двойная кавычка	'\u005C'
'\''	одинарная кавычка	'\u0027'
'\\'	обратная косая черта	'\u0022'

Преобразования целочисленных типов



	Допускается неявное преобразование между целочисленными типами при условии что не может быть потеряна информация. В противном случае допускается явное преобразование. Для явного преобразования необходимо указать тип к которому происходит преобразование.

Преобразование типов

```
public class ByteIntConversionDemo {

   public static void main(String[] args) {

      int xInt;
      byte xByte;

      System.out.println("Implicit conversion byte to int ...");

      xByte = 100;
      xInt = xByte;

      System.out.println("xByte variable's value: " + xByte);
      System.out.println("xInt variable's value: " + xInt);

      System.out.println("\nNow explicit conversion int to byte ...");

      xInt = 150;
      xByte = (byte) xInt;

      System.out.println("xInt variable's value: " + xInt);
      System.out.println("xByte variable's value: " + xByte);
   }
}
```

```
Implicit conversion byte to int ...
xByte variable's value: 100
xInt variable's value: 100

Now explicit conversion int to byte ...
xInt variable's value: 150
xByte variable's value: -106
```

Арифметические операторы

Оператор	Результат
+	Сложение (унарный плюс)
-	Вычитание (унарный минус)
*	Умножение
/	Деление
%	Модуль от деления
++	Инкремент
+=	Сложение с присваиванием
-=	Вычитание с присваиванием
*=	Умножение с присваиванием
/=	Деление с присваиванием
%=	Модуль от деления с присваиванием
--	Декремент

Если операнды целочисленных типов кроме long то они все приводятся к int и результат тоже будет int. Если операнды целочисленных типов и один из них long то результат будет long. Переполнение целочисленных типов маскируется.

Результат арифметических операций int

```
public class ByteDemo {

   public static void main(String[] args) {

      byte a = 10;
      byte b = 20;


      byte c = (byte) (a + b);

      System.out.println("And the result is: " + c);
   }
}
```

```
And the result is: 30
```

Маскировка переполнения целых чисел

```
public class SilentOverflowDemo {

   public static void main(String[] args) {

      int posint = 2147483647;

      System.out.println("number is: " + posint);
      System.out.println("number + 1 is: " + (posint + 1));

      System.out.println();

      int negint = -2147483648;

      System.out.println("number is: " + negint);
      System.out.println("number - 1 is: " + (negint - 1));
   }
}
```

```
number is: 2147483647
number + 1 is: -2147483648

number is: -2147483648
number - 1 is: 2147483647
```

Маскировка переполнения целых чисел

```
public class SilentOverflowDemo {

   public static void main(String[] args) {

      int posint = 2147483647;

      System.out.println("number is: " + posint);
      System.out.println("number + 1 is: " + (posint + 1));

      System.out.println();

      int negint = -2147483648;

      System.out.println("number is: " + negint);
      System.out.println("number - 1 is: " + (negint - 1));
   }
}
```

```
number is: 2147483647
number + 1 is: -2147483648

number is: -2147483648
number - 1 is: 2147483647
```

Маскировка переполнения целых чисел

```
public class SilentOverflowDemo {

   public static void main(String[] args) {

      int posint = 2147483647;

      System.out.println("number is: " + posint);
      System.out.println("number + 1 is: " + (posint + 1));

      System.out.println();

      int negint = -2147483648;

      System.out.println("number is: " + negint);
      System.out.println("number - 1 is: " + (negint - 1));
   }
}
```

```
number is: 2147483647
number + 1 is: -2147483648

number is: -2147483648
number - 1 is: 2147483647
```

Маскировка переполнения целых чисел

```
public class SilentOverflowDemo {

   public static void main(String[] args) {

      int posint = 2147483647;

      System.out.println("number is: " + posint);
      System.out.println("number + 1 is: " + (posint + 1));

      System.out.println();

      int negint = -2147483648;

      System.out.println("number is: " + negint);
      System.out.println("number - 1 is: " + (negint - 1));
   }
}
```

```
number is: 2147483647
number + 1 is: -2147483648

number is: -2147483648
number - 1 is: 2147483647
```

Маскировка переполнения целых чисел

```
public class SilentOverflowDemo {

   public static void main(String[] args) {

      int posint = 2147483647;

      System.out.println("number is: " + posint);
      System.out.println("number + 1 is: " + (posint + 1));

      System.out.println();

      int negint = -2147483648;

      System.out.println("number is: " + negint);
      System.out.println("number - 1 is: " + (negint - 1));
   }
}
```

```
number is: 2147483647
number + 1 is: -2147483648

number is: -2147483648
number - 1 is: 2147483647
```

Результат арифметических операций int или long

```
public class ResultOverflowDemo {

   public static void main(String[] args) {

      long yearMilliseconds = 1000 * 365 * 24 * 60 * 60;
      System.out.println("Wrong number of milliseconds per year: " + yearMilliseconds);

      yearMilliseconds = 1000L * 365 * 24 * 60 * 60;
      System.out.println("Correct number of milliseconds per year: " + yearMilliseconds);

   }
}
```

```
Wrong number of milliseconds per year: 1471228928
Correct number of milliseconds per year: 31536000000
```

Результат арифметических операций int или long

```
public class ResultOverflowVarDemo {

   public static void main(String[] args) {

      int millis = 1000;
      int days = 365;
      int hours = 24;
      int minutes = 60;
      int seconds = 60;

      long yearMilliseconds = (long)(millis * days * hours * minutes * seconds);
      System.out.println("Wrong conversion int to long variable : " + yearMilliseconds);

      yearMilliseconds = (long)(millis) * days * hours * minutes * seconds;
      System.out.println("Correct conversion int to long variable : " + yearMilliseconds);
   }
}
```

```
Wrong conversion int to long variable : 1471228928
Correct conversion int to long variable : 31536000000
```

Деление на ноль

```
public class ArithmeticExceptionDemo {

   public static void main(String[] args) {

      int a = 10;
      int b = 0;

      System.out.println("We will try to divide " + a + " by " + b);
      System.out.println(a / b);

   }
}
```

```
We will try to divide 10 by 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
at primitives.ArithmeticExceptionDemo.main(ArithmeticExceptionDemo.java:11)
```

Операторы сравнения и упорядоченности

Оператор	Результат
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Операнды могут быть любых целочисленных типов. Результатом операции сравнения или упорядоченности всегда будет boolean.

Битовые целочисленные операторы

Оператор	Результат
~	Битовый унарный не
&	Битовый и
|	Битовый или
^	Битовый исключающий или (XOR)
>>	Сдвиг вправо
>>>	Сдвиг вправо с заполнением нулями
<<	Сдвиг влево
&=	Битовый и с присваиванием
|=	Битовый или с присваиванием
^=	Битовый ислючающий или с присваиванием
>>=	Сдвиг вправо с присваиванием
>>>=	Сдвиг вправо с присваиванием с заполнением нулями

Вещественные типы

Вещественные литералы

Тип	Суффикс	Пример	Пример
float	F или f	-123.412F	-1.23412e+2F
double	D или d	123412.34123D	1.2341234123e+5D

Вещественные литералы без суффикса (по умолчанию) или с суффиксом D или d имеют тип double и могут использоваться для инициализации переменных типа double. Вещественные литералы с суффиксом F или f имеют тип float и могут быть использованы для инициализации переменных типа float или типа double. Значение литерала должно быть в соответствующем диапазоне иначе произойдёт ошибка компиляции.

Максимальные и минимальные по модулю значения

Тип	Размер	Минимальные по модулю значения	Максимальные по модулю значения
float	32 бита	$2^{-149} \approx \pm 1.4012984643248170707e\text{-}45$	$(2-2^{-23})\cdot 2^{127} \approx \pm 3.4028234663852886e+38$
double	64 бита	$2^{-1074} \approx \pm 4.9406564584124654e\text{-}324$	$(2-2^{-52})\cdot 2^{1023} \approx \pm 1.7976931348623157e+308$

Специальные значения

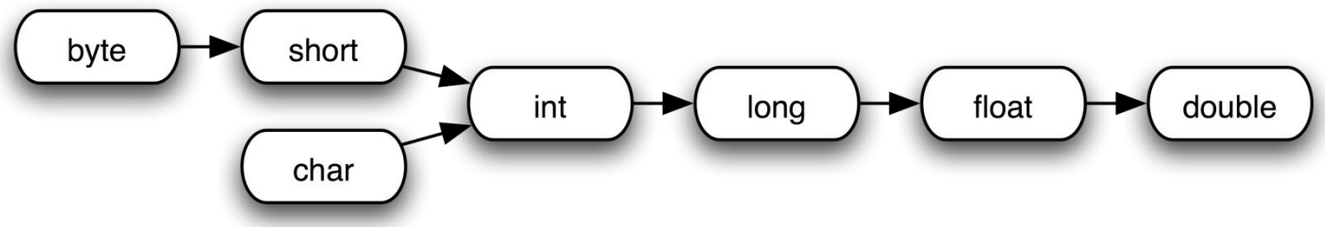
Значение	Размер
Infinity	Положительное значение по модулю больше максимального значения по модулю
-Infinity	Отрицательное значение по модулю больше максимального значения по модулю
0	Положительное значение по модулю меньше минимального значения по модулю
-0	Отрицательное значение по модулю меньше минимального значения по модулю
NaN	Не число - результат операций не имеющих смысла

Специальные значения

```
public class SpecialValuesDemo {

   public static void main(String[] args) {

      double max = Double.MAX_VALUE;
      double min = Double.MIN_VALUE;

      System.out.println("Maximum double value approximately is: " + max);
      System.out.println("Minimum positive double value approximately is: " + min);
      System.out.println("Positive infinity is: " + max * 2);
      System.out.println("Positive zero is: " + min / 2);
      System.out.println("Negative infinity is: " + (-max * 2));
      System.out.println("Negative zero is: " + (-min / 2));

      System.out.println("Not a number: " + Math.sqrt(-1));
   }
}
```

```
Maximum double value approximately is: 1.7976931348623157E308
Minimum positive double value approximately is: 4.9E-324
Positive infinity is: Infinity
Positive zero is: 0.0
Negative infinity is: -Infinity
Negative zero is: -0.0
Not a number: NaN
```

Преобразования



	Допускается неявное преобразование float в double и явное преобразование double в float. Допускается неявное преобразование любого целочисленного типа в float и double и явное преобразование float и double в целочисленные типы. Неявное преобразование long в float и double, а также неявное преобразование int в float могут привести к потере точности.

Преобразование типов

```
public class IntDoubleConversionDemo {

   public static void main(String[] args) {

      int xInt;
      double xDouble;

      System.out.println("Implicit conversion int to double ...");

      xInt = 120;
      xDouble = xInt;

      System.out.println("xInt variable's value: " + xInt);
      System.out.println("xDouble variable's value: " + xDouble);

      System.out.println("\nNow explicit conversion double to int ...");

      xDouble = 3.8547;
      xInt = (int) xDouble;

      System.out.println("xDouble variable's value: " + xDouble);
      System.out.println("xInt variable's value: " + xInt);

   }
}
```

```
Implicit conversion int to double ...
xInt variable's value: 120
xDouble variable's value: 120.0

Now explicit conversion double to int ...
xDouble variable's value: 3.8547
xInt variable's value: 3
```

Арифметические операторы

Оператор	Результат
+	Сложение (унарный плюс)
-	Вычитание (унарный минус)
*	Умножение
/	Деление
%	Модуль от деления
++	Инкремент
+=	Сложение с присваиванием
-=	Вычитание с присваиванием
*=	Умножение с присваиванием
/=	Деление с присваиванием
%=	Модуль от деления с присваиванием
--	Декремент

Операция	Результат
Math.sqrt(-1.0)	NaN
0.0 / 0.0	NaN
1.0 / 0.0	Infinity
-1.0 / 0.0	-Infinity
NaN + 1.0	NaN
Infinity + 1.0	Infinity
Infinity + Infinity	Infinity

Вещественная арифметика как и хранение вещественных чисел не является точной. Для точных финансовых вычислений необходимо использовать тип BigDecimal.

Если оба операнда вещественные и хотя бы один из них double то результат будет double в противном случае результат будет float. Если один операнд вещественный а другой целочисленный то целочисленный операнд будет приведён к типу вещественного операнда и результат будет типа вещественного операнда.

Неточность вещественной арифметики

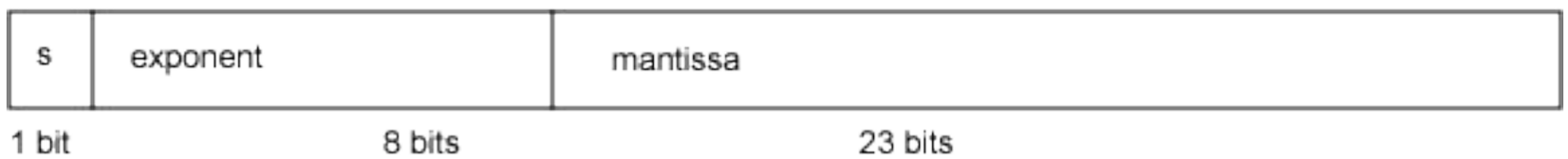
```
public class ImprecisionDemo {

   public static void main(String[] args) {

      System.out.println(0.1);
      System.out.println(0.1 + 0.1);
      System.out.println(0.1 + 0.1 + 0.1);

   }
}
```

```
0.1
0.2
0.30000000000000004
```

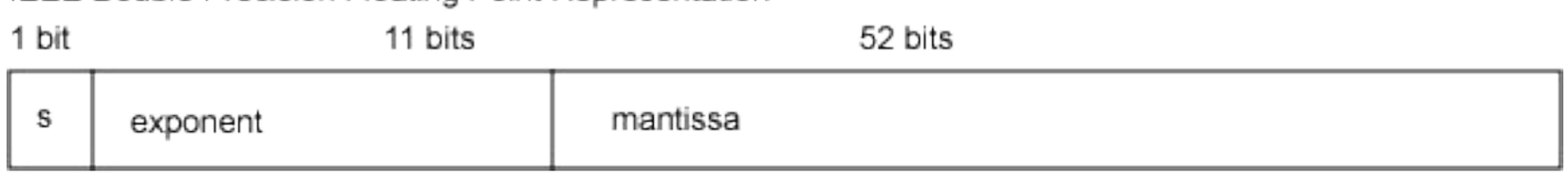
Такого вещественного значения как 0.1 не существует. Это связано с тем как вещественные числа float и double представляются в памяти. Кроме того арифметика не является точной.

Как хранятся числа с плавающей точкой

IEEE Floating Point Representation



IEEE Double Precision Floating Point Representation



Пример хранения float 0.8125 = 0.5+0.25+0.0625

$$(-1)^s \times (1+m) \times 2^{(e - 127)}$$

$0.8125 = (-1)^0 \times (1 + 0.5 + 0.125) \times 2^{-1}$

bits:	**31**	**30-23**	**22-0**
binary:	**0**	**01111110**	**10100000000000000000000**
decimal:	**0**	**126**	**5242880**

$2^{e-127}(1 + m / 2^{23}) =$
$2^{-1}(1 + (0.5 + 0.125)) =$
$2^{-1}(1 + 5242880/8388608) =$
$2^{-1}(1 + 0.625) = 0.8125$

Значение разряда в бинарной записи может быть 0 или 1. Первая 1 не хранится в мантиссе так как значение первого разряда всегда 1.

Пример хранения float 0.085

$$(-1)^s \times (1+m) \times 2^{(e - 127)}$$

```
0.085:
bits:    31   30-23      22-0
binary:   0   01111011   01011100001010001111011
decimal:  0   123        3019899
```

$2^{e-127}(1 + m / 2^{23}) =$
$2^{-4}(1 + (0.25 + 0.0625 + 0.03125 + ...)) =$
$2^{-4}(1 + 3019899/8388608) =$
$11408507/134217728 =$
$0.085000000894069671630859375$

Операторы сравнения и упорядоченности

Оператор	Результат
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

Операция	Результат
NaN > 1.0	false
NaN == 1.0	false
NaN < 1.0	false
NaN == NaN	false
0.0 == -0.0	true

Операнды могут быть любых численных типов. Результатом операции сравнения или упорядоченности всегда будет boolean.

Операторы сравнения и неточность вещественной арифметики

```
public class NeverEndingDemo {

   public static void main(String[] args) throws InterruptedException{

     for (double x = 0; x != 10; x += 0.1) {

        System.out.println(x);
        Thread.sleep(1000);
     }
   }
}
```

```
9.099999999999984
9.199999999999983
9.299999999999983
9.399999999999983
9.499999999999982
9.599999999999982
9.699999999999982
9.799999999999981
9.89999999999998
9.99999999999998
10.09999999999998
10.19999999999998
10.29999999999998
10.399999999999979
10.499999999999979
10.599999999999978
...
```