

8.5

Точность вычислений

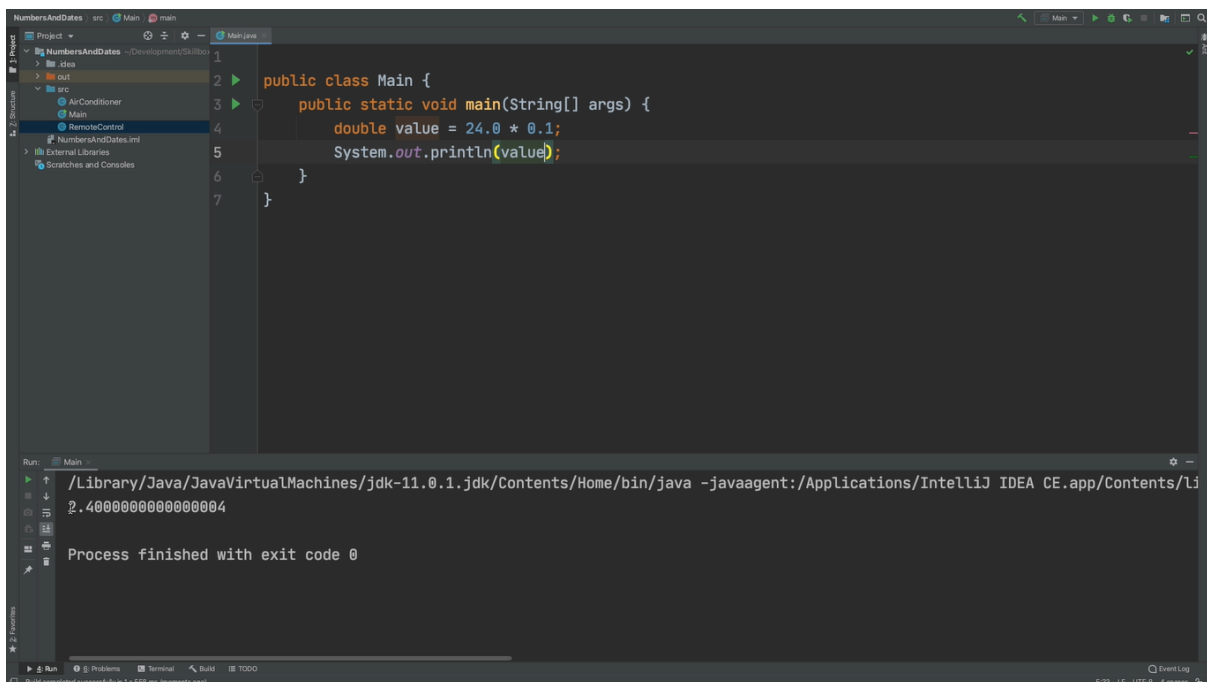
00:00–01:30

Введение

Привет!

Поговорим о проблеме точности вычислений при использовании чисел с плавающей точкой, о причинах этой проблемы и о некоторых важных моментах, которые необходимо в связи с ней учитывать.

Бывают ситуации, при которых результат вычисления совершенно не соответствует нашим ожиданиям. Например, умножаем число типа `double` 24 и 0,1. По идее, должен быть результат 2,4, но результат будет немного другим. Запускаем и видим результат:



```
1 public class Main {
2
3     public static void main(String[] args) {
4         double value = 24.0 * 0.1;
5         System.out.println(value);
6     }
7 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib

2.4000000000000004

Process finished with exit code 0

Другой пример с числом типа `float`:

```
1 public class Main {
2     public static void main(String[] args) {
3         float value = 0.3f + 0.4f;
4         System.out.println(value);
5     }
6 }
7 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
0.70000005

Process finished with exit code 0

Попробуем разобраться, почему так происходит. Сначала вспомним, в каком виде числа с плавающей точкой хранятся в памяти:

Точность вычислений при использовании чисел “float” и “double”

	Знак	Порядок	Мантисса
float 32 bit	1 bit	8 bit	23 bit
double 64 bit	1 bit	11 bit	52 bit

$$- 5.34234 \times 10^{17}$$

Числа с плавающей точкой хранятся в памяти в виде трёх частей: знака, порядка и мантиссы. Каждая из этих частей представляет собой набор полей и единиц, то есть хранится в двоичной системе счисления.

01:30–02:52

Особенности систем счислений

Не будем погружаться в подробности, уточним только, что в каждой системе счисления есть определённые ограничения на то, как в них могут быть представлены числа с плавающей точкой.

К примеру, в десятичной системе счисления нельзя представить результат деления 1 на 3, потому что результат — бесконечность. Это специальная запись, которая не реализуется в этой системе. Точно так же в двоичной системе счисления нельзя точно представить результат деления 1 на 10. Пример такого деления вы можете увидеть ниже:

Деление 1 на 10 в двоичной системе

В двоичной системе:

$1 / 1010 = 0\text{b}0,00011001100110011001100110011001100110011001100110$

Если перевести результат в десятичную систему:

$0,099999999999999964472863211994990706443786621093750$

С помощью нолей и единиц записать точно такое число не получится. Именно эта особенность двоичной системы счисления делает невозможными точное вычисление с использованием float и double.

02:52–07:22

Важные моменты

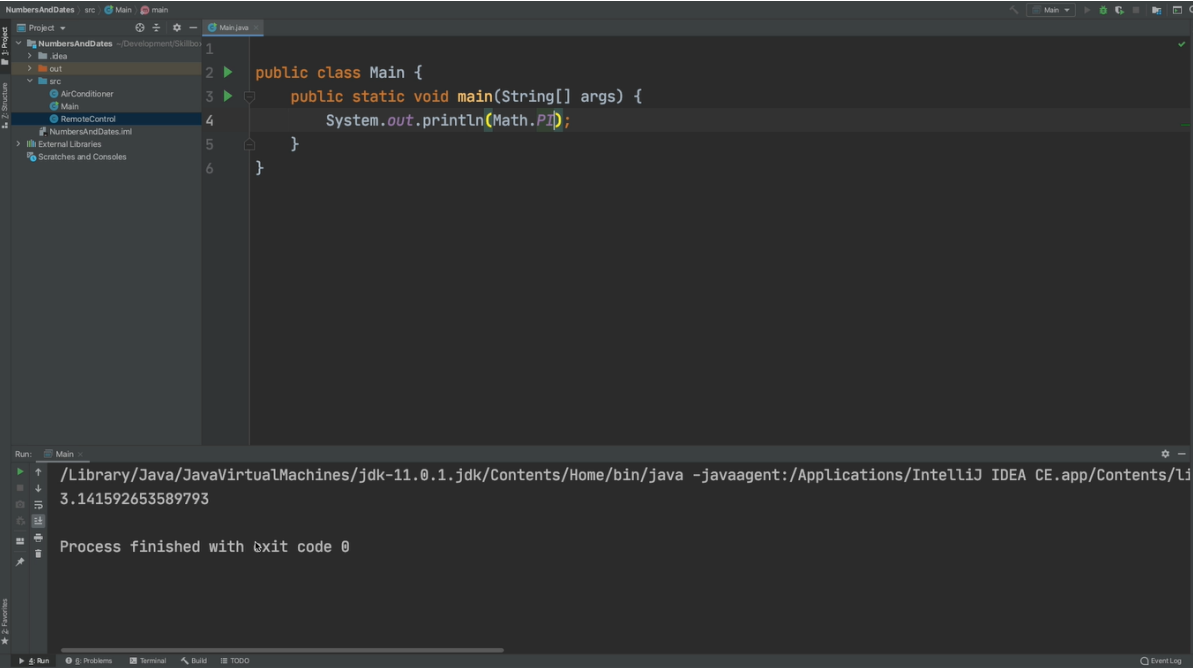
Три важных момента при работе с числами с плавающей точкой, которые связаны с точностью.

Первый важный момент: числа double точнее чисел float. Это связано с тем, что число double занимает в памяти в два раза больше места, чем число float, и цифр после запятой в числе double может быть значительно больше:

Хранение в памяти чисел “float” и “double”

	Знак	Порядок	Мантисса
float 32 bit	1 bit	8 bit	23 bit
double 64 bit	1 bit	11 bit	52 bit

Посмотрим на примере. Если возьмём число π , то увидим, что это число имеет 15 знаков после точки:



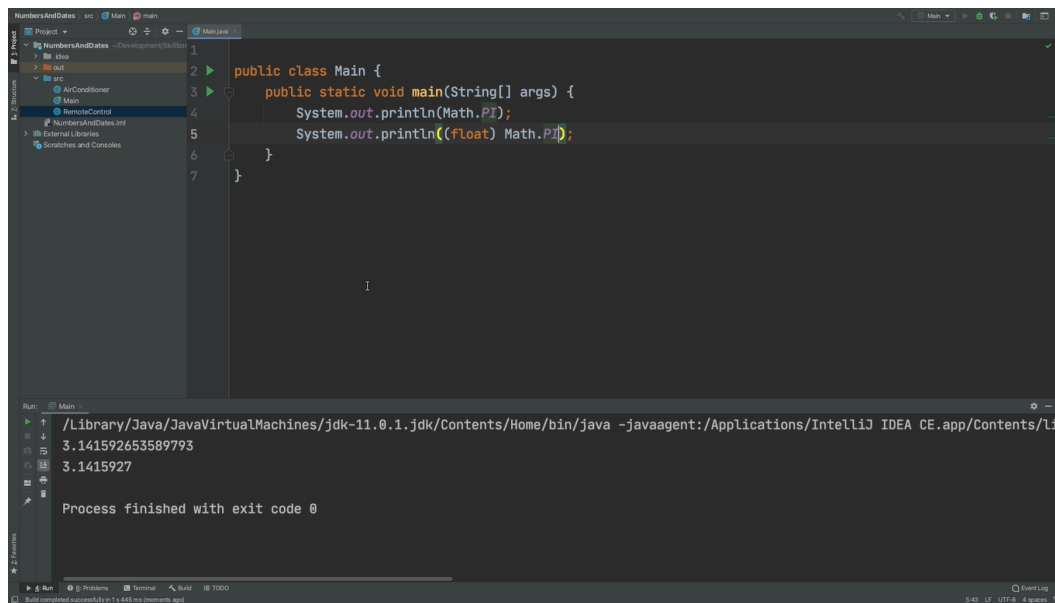
```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println(Math.PI);
4     }
5 }
6 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
3.141592653589793

Process finished with `exit` code 0

Если мы преобразуем это число во float, то количество знаков будет гораздо меньше — их будет всего семь.



```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println(Math.PI);
4         System.out.println((float) Math.PI);
5     }
6 }
7
```

Run: Main

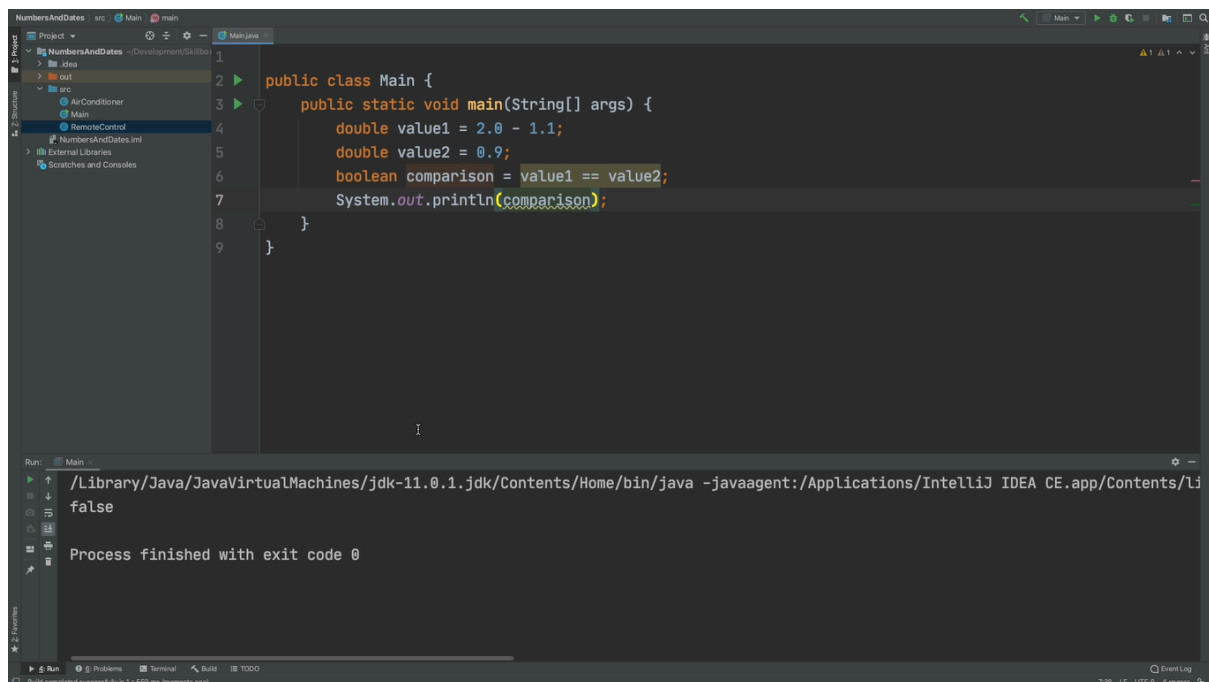
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
3.141592653589793
3.1415927

Process finished with exit code 0

Второй важный момент: при работе с числами с плавающей точкой нельзя использовать точное сравнение.

Вот очень простой пример, когда такое сравнение работать не будет из-за проблем с точностью.

Есть некоторое число $value1 = 2.0 - 1.1$ и второе целое число $value2 = 0.9$. Если мы сравним эти числа, то увидим, что это сравнение будет равно false:



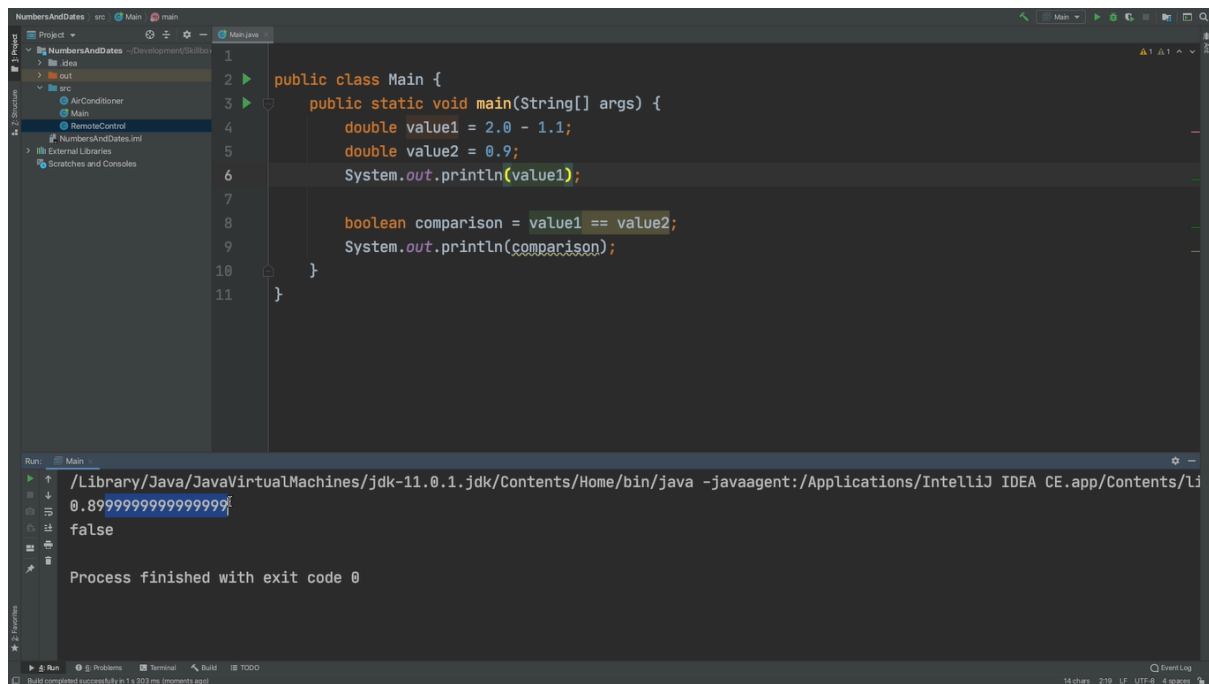
```
1 public class Main {
2     public static void main(String[] args) {
3         double value1 = 2.0 - 1.1;
4         double value2 = 0.9;
5         boolean comparison = value1 == value2;
6         System.out.println(comparison);
7     }
8 }
9
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
false

Process finished with exit code 0

Почему? Потому что результат первого вычисления не будет точно равен 0,9.



```
1 public class Main {
2     public static void main(String[] args) {
3         double value1 = 2.0 - 1.1;
4         double value2 = 0.9;
5         System.out.println(value1);
6
7         boolean comparison = value1 == value2;
8         System.out.println(comparison);
9     }
10 }
11 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
0.8999999999999999
false

Process finished with exit code 0

Что делать? Чтобы сравнивать такие числа, можно использовать некий параметр, содержащий в себе границу точности:



```
1 public class Main {
2     public static void main(String[] args) {
3
4         double epsilon = 0.000000001;
5
6         double value1 = 2.0 - 1.1;
7         double value2 = 0.9;
8         boolean comparison = Math.abs(value1 - value2) < epsilon;
9         System.out.println(comparison);
10     }
11 }
12 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
true

Process finished with exit code 0

Если погрешность вычисления меньше, чем заданный параметр, тогда можно считать эти значения равными. Рекомендуется также вынести этот параметр в константу, чтобы во всём проекте использовать одну и ту же точность.



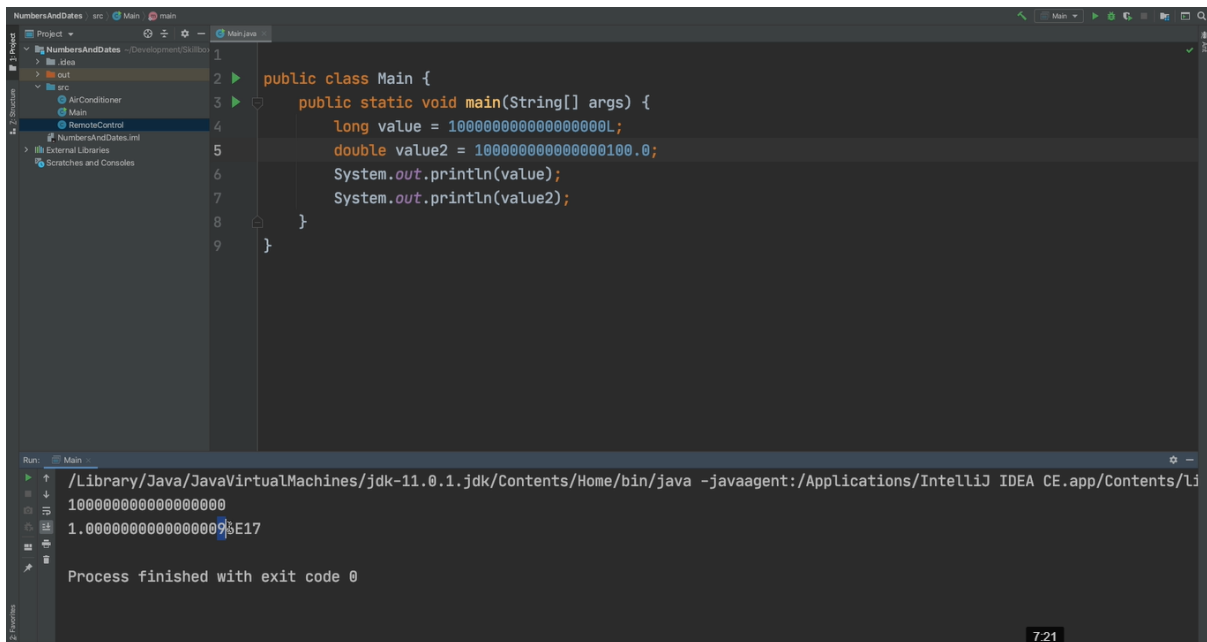
```
1 public class Main {
2
3
4     public static final double EPSILON = 0.00000001;
5
6     public static void main(String[] args) {
7
8
9
10        double value1 = 2.0 - 1.1;
11        double value2 = 0.9;
12        boolean comparison = Math.abs(value1 - value2) < EPSILON;
13        System.out.println(comparison);
14    }
15 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
true

Process finished with exit code 0

Третий важный момент: большие числа с плавающей точкой тоже могут быть неточными. Помните, что несмотря на то, что числа с плавающей точкой имеют одну и ту же размерность с некоторыми целыми числами, в них хранятся совершенно разные числа. Например:



```
1 public class Main {
2
3     public static void main(String[] args) {
4         long value = 100000000000000000L;
5         double value2 = 100000000000000000.0;
6         System.out.println(value);
7         System.out.println(value2);
8     }
9 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
100000000000000000
1.0000000000000000E17

Process finished with exit code 0

07:22–07:49

Выводы

Итак, в этой теме вы познакомились с проблемой точности чисел с плавающей точкой и узнали несколько важных моментов, о которых необходимо помнить в связи с ней.

В следующей теме вы узнаете о специальных классах, которые, в отличие от чисел с плавающей точкой, позволяют производить не только точные вычисления, но и вычисления с очень большими числами.