

## 8.6

# Работа с большими числами и точными числами

00:00–00:25

## Введение

Привет!

В предыдущей теме вы познакомились с проблемой неточности чисел с плавающей точкой. Кроме того, размер классических чисел, с которым вы уже знакомы, ограничен, а нам могут потребоваться числа гораздо больше.

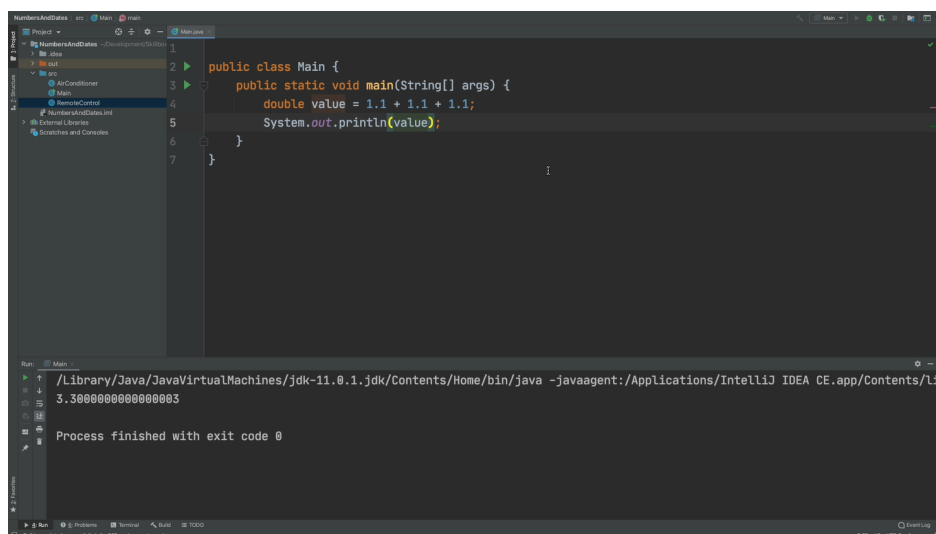
Давайте познакомимся со специальными классами, позволяющими производить не только максимально точные вычисления, но и работать с очень большими числами.

00:25–02:29

## Проблемы при работе с «обычными» числами

Проблемы с классическими числами, с которыми вы уже знакомы.

**Первая проблема — неточность** вычислений при работе с числами с плавающей точкой. Если вы попытаетесь посчитать что-нибудь подобное, то результат будет неточным:



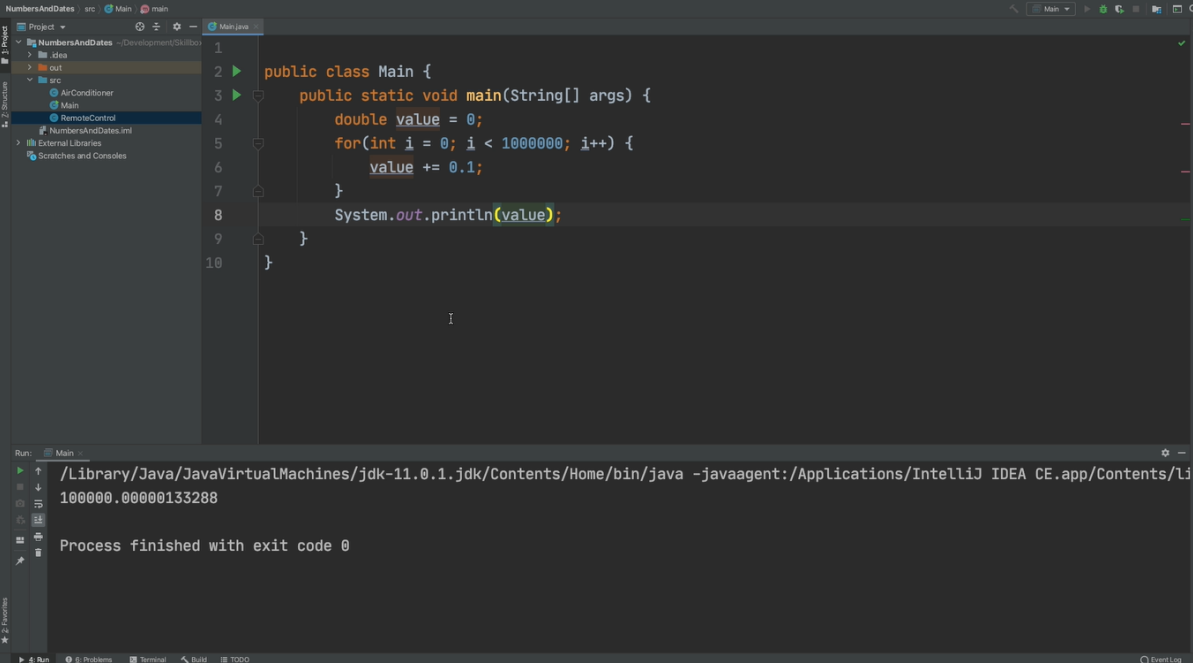
```
1 public class Main {
2     public static void main(String[] args) {
3         double value = 1.1 + 1.1 + 1.1;
4         System.out.println(value);
5     }
6 }
7
```

Run: Main

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea-runtime.jar -jar /Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea-runtime.jar 3.3000000000000003
```

Process finished with exit code 0

Такие неточности имеют свойство накапливаться, и если вы работаете с точными научными вычислениями или с финансами, то они могут быть очень существенны для конечного результата. Ещё один пример простого вычисления с большой погрешностью:



```
1 public class Main {
2     public static void main(String[] args) {
3         double value = 0;
4         for(int i = 0; i < 1000000; i++) {
5             value += 0.1;
6         }
7         System.out.println(value);
8     }
9 }
10 }
```

Run: Main

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Lj
100000.00000133288
```

Process finished with exit code 0

Здесь ошибка будет очень грубой. При точных научных расчётах такие неточности недопустимы.

**Вторая проблема** классических чисел — это **ограниченность их значений**. Если не рассчитать, например, умножить триллион на триллион, то можно получить классическое переполнение.



```
1 public class Main {
2     public static void main(String[] args) {
3         long value = 1_000_000_000_000L;
4         System.out.println(value * value);
5     }
6 }
7
8 }
```

Run: Main

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
2003764205206896640
Process finished with exit code 0
```

Такие ошибки категорически недопустимы в определённых областях.

02:29–09:14

## Специальные классы

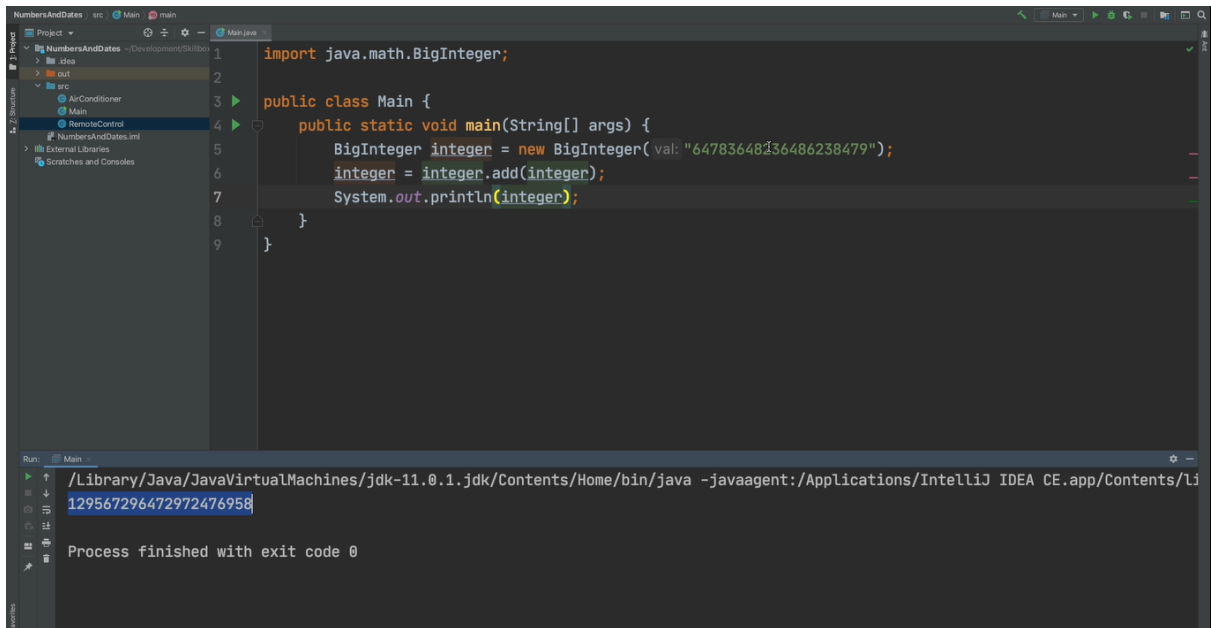
В Java есть специальные классы, которые позволяют производить точные вычисления, в том числе с очень большими числами.

**Первый** такой класс — **BigInteger**. Из названия следует, что этот класс предназначен для работы с целыми числами.

Посмотрим, как он работает.

Объект класса **BigInteger** можно создавать как на основе числа, так и на основе строки.

Давайте создадим на основе строки. У класса **BigInteger** есть множество методов, которые позволяют производить практически любые математические операции с этими числами. Например, операция сложения **add**.



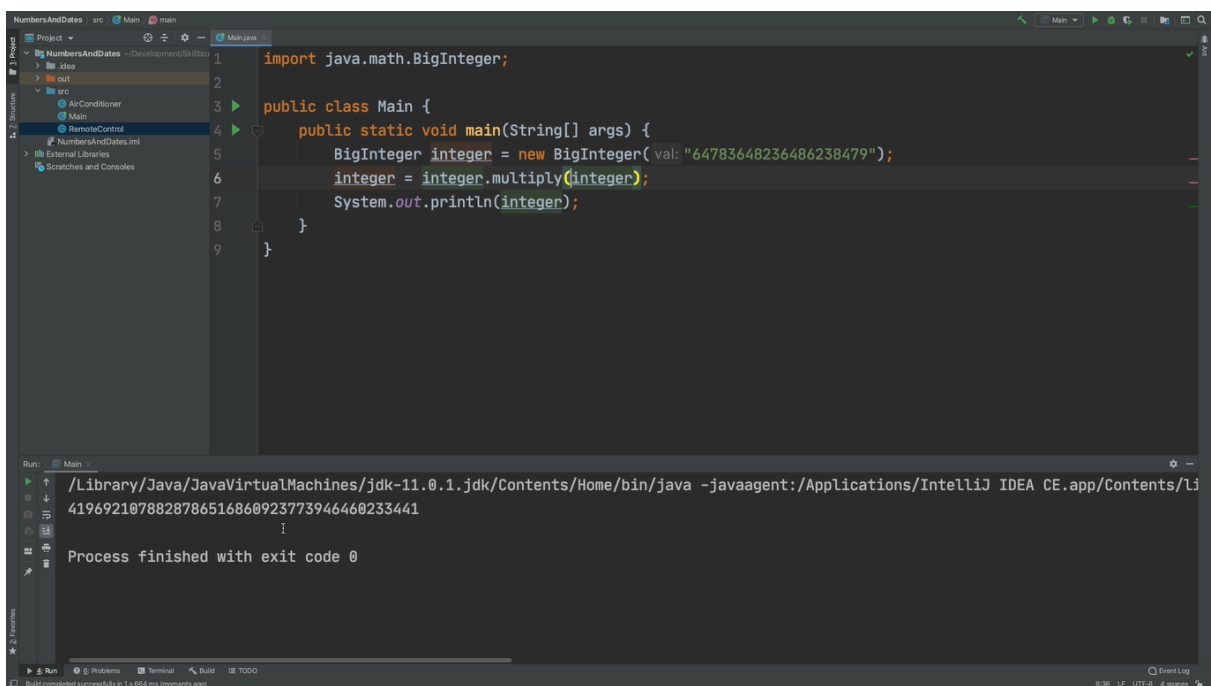
The screenshot shows the IntelliJ IDEA IDE with a project named "NumbersAndDates". The "Main.java" file is open, displaying the following code:

```
1 import java.math.BigInteger;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigInteger integer = new BigInteger(val: "64783648236486238479");
6         integer = integer.add(integer);
7         System.out.println(integer);
8     }
9 }
```

The Run window at the bottom shows the output of the program:

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
129567296472972476958
Process finished with exit code 0
```

Можно взять и умножить один такой большой integer на другой такой же большой integer с помощью метода **multiply**:



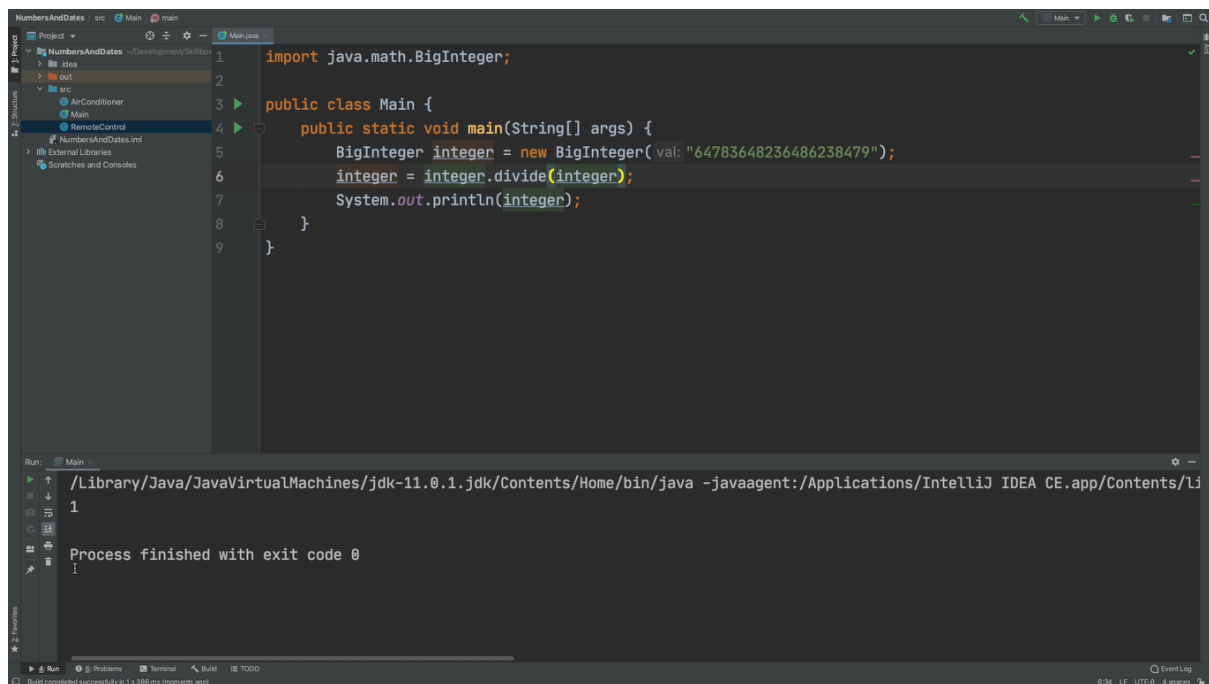
The screenshot shows the IntelliJ IDEA IDE with the same project "NumbersAndDates". The "Main.java" file is open, displaying the following code:

```
1 import java.math.BigInteger;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigInteger integer = new BigInteger(val: "64783648236486238479");
6         integer = integer.multiply(integer);
7         System.out.println(integer);
8     }
9 }
```

The Run window at the bottom shows the output of the program:

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
4196921078828786516860923773946460233441
Process finished with exit code 0
```

Также есть операция деления — **divide**.



```
1 import java.math.BigInteger;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigInteger integer = new BigInteger(val: "64783648236486238479");
6         integer = integer.divide(integer);
7         System.out.println(integer);
8     }
9 }
```

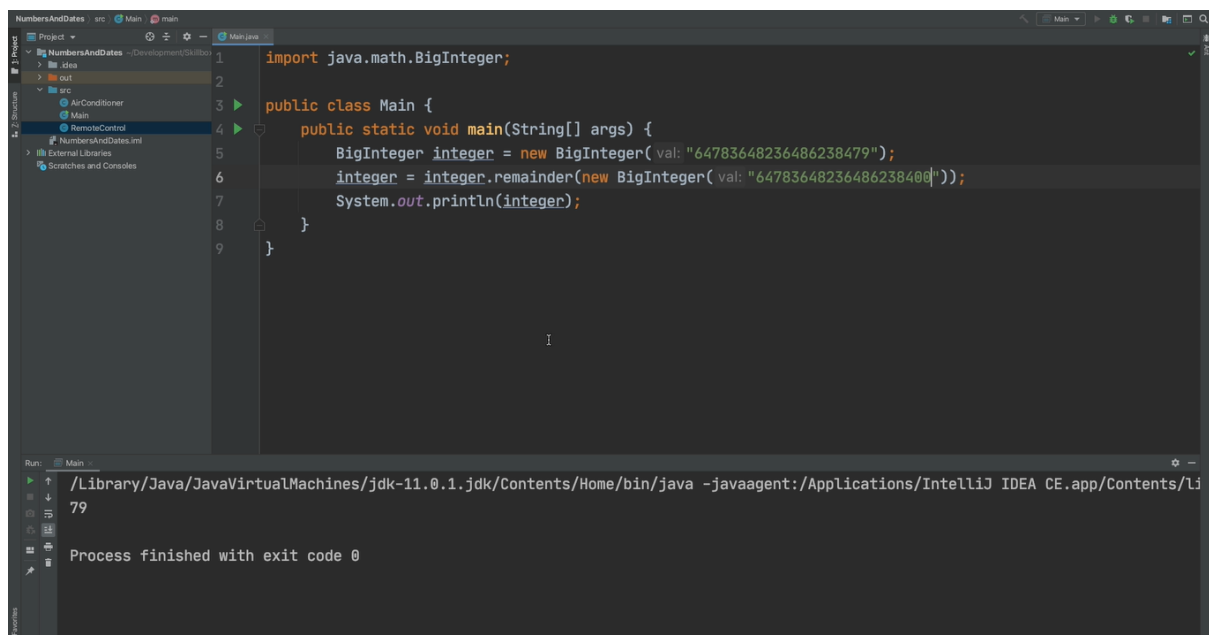
Run: Main

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

1

Process finished with exit code 0

Ещё здесь есть операция расчёта остатка от деления — это метод **remainder**:



```
1 import java.math.BigInteger;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigInteger integer = new BigInteger(val: "64783648236486238479");
6         integer = integer.remainder(new BigInteger(val: "64783648236486238400"));
7         System.out.println(integer);
8     }
9 }
```

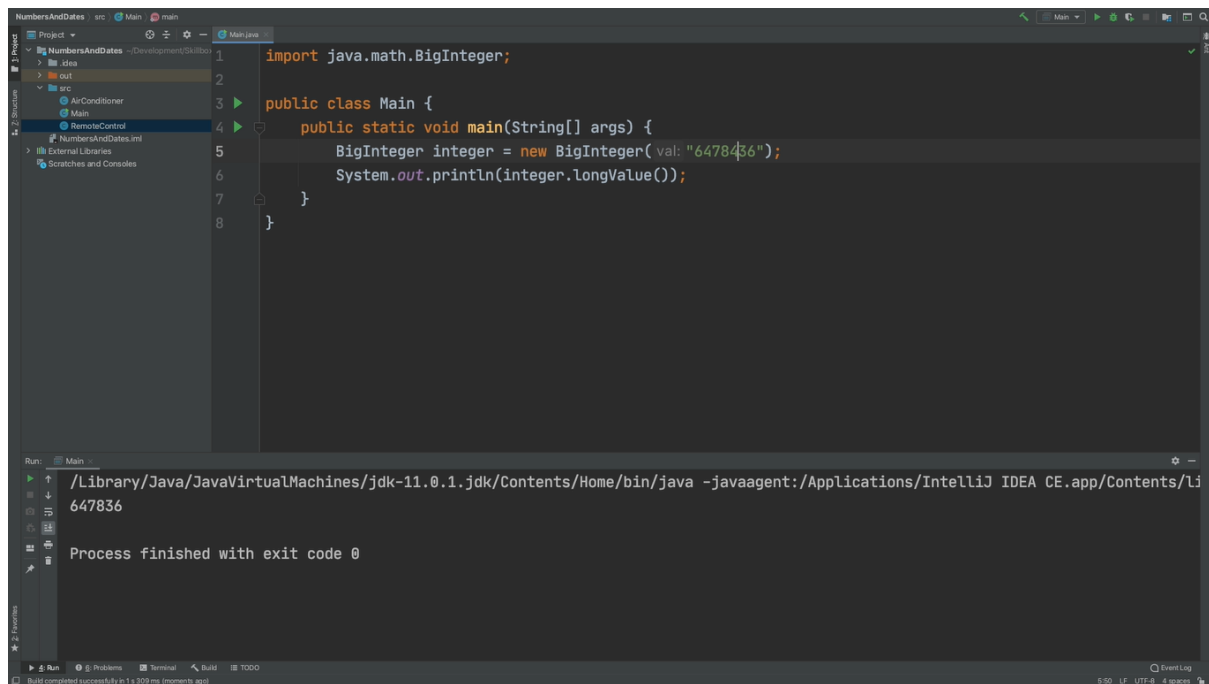
Run: Main

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

79

Process finished with exit code 0

Числа BigInteger можно превращать в числа типа int и long с помощью метода long value или int value:

The screenshot shows an IDE window with a project named 'NumbersAndDates'. The 'src' folder contains a 'Main' class. The code in 'Main.java' is as follows:

```
1 import java.math.BigInteger;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigInteger integer = new BigInteger( val: "6478436");
6         System.out.println(integer.longValue());
7     }
8 }
```

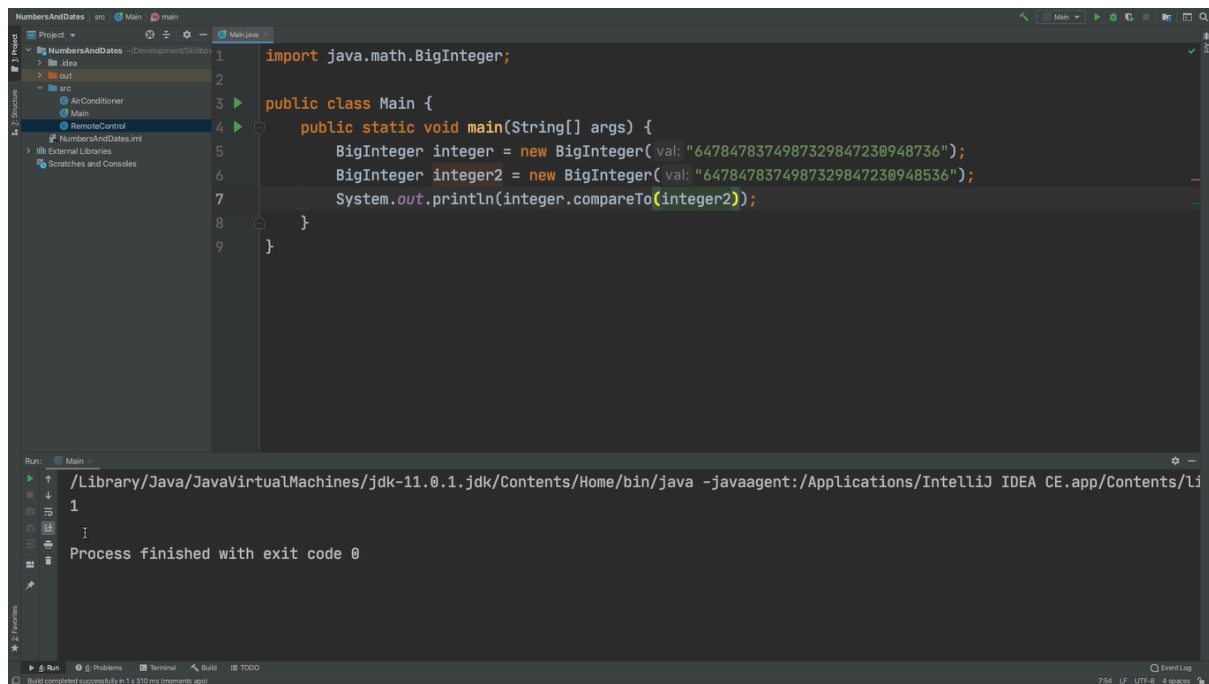
The 'Run' tab at the bottom shows the execution path: `/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Lj`. The output is `647836`, and the process finished with exit code 0.

Важно помнить, что число может в эти типы не влезть.

У класса `BigInteger` есть метод `compare`, который позволяет сравнивать числа.

Результат может быть отрицательным, нулевым или положительным.

- Если результат положительный, значит число, у которого вызывается метод, больше, чем число, которое передано в качестве параметра.
- Если результат отрицательный, значит число, у которого вызывается метод, меньше, чем число, которое передано в качестве параметра.
- Если эти числа равны, то результат будет равен нулю.



```
1 import java.math.BigInteger;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigInteger integer = new BigInteger(val: "6478478374987329847230948736");
6         BigInteger integer2 = new BigInteger(val: "6478478374987329847230948536");
7         System.out.println(integer.compareTo(integer2));
8     }
9 }
```

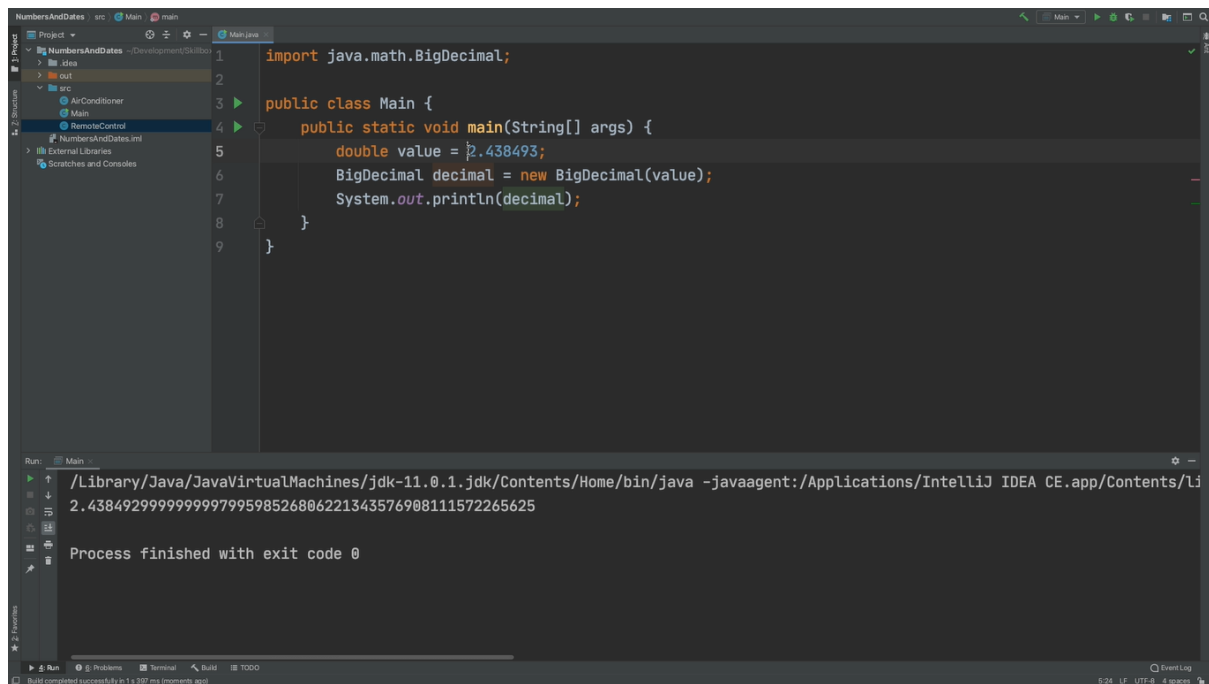
Run: Main

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li
1
I
Process finished with exit code 0
```

Главное, что размер этих чисел может быть ограничен только размером оперативной памяти вашего компьютера, то есть вы можете возвести их в любую степень. Вычисления займут какое-то время и, возможно, несколько экранов в консоли, но они будут точными.

**Второй такой класс — `BigDecimal`.** Он предназначен для работы с числами с плавающей точкой.

Посмотрим, как он работает. Если это число создавать из числа `double` или типа `float`, то могут возникнуть проблемы с точностью. Выведем это число в консоль и увидим, что оно совсем не равно тому числу, которое было передано ему в параметры.



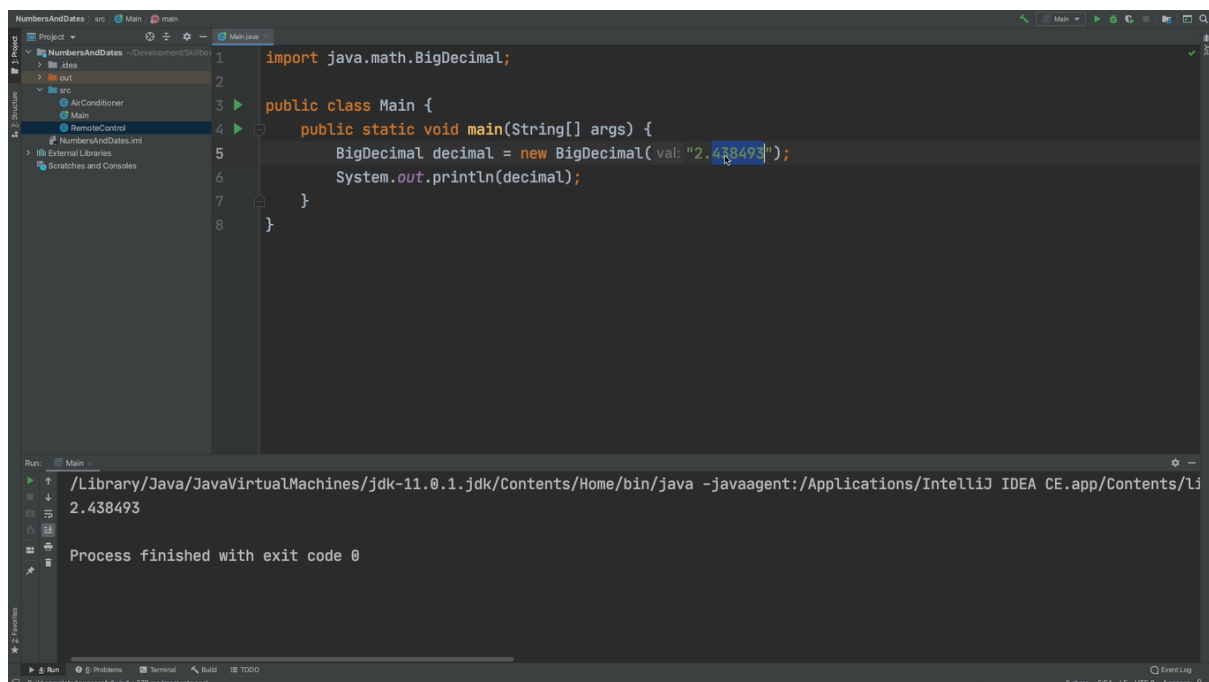
```
1 import java.math.BigDecimal;
2
3 public class Main {
4     public static void main(String[] args) {
5         double value = 2.438493;
6         BigDecimal decimal = new BigDecimal(value);
7         System.out.println(decimal);
8     }
9 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li  
2.438492999999999799598526806221343576908111572265625

Process finished with exit code 0

Чтобы задавать такие числа точно, их нужно задавать на основе строки. Тогда результат будет ровно таким же, какая передана строка:



```
1 import java.math.BigDecimal;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigDecimal decimal = new BigDecimal(val: "2.438493");
6         System.out.println(decimal);
7     }
8 }
```

Run: Main

/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Li  
2.438493

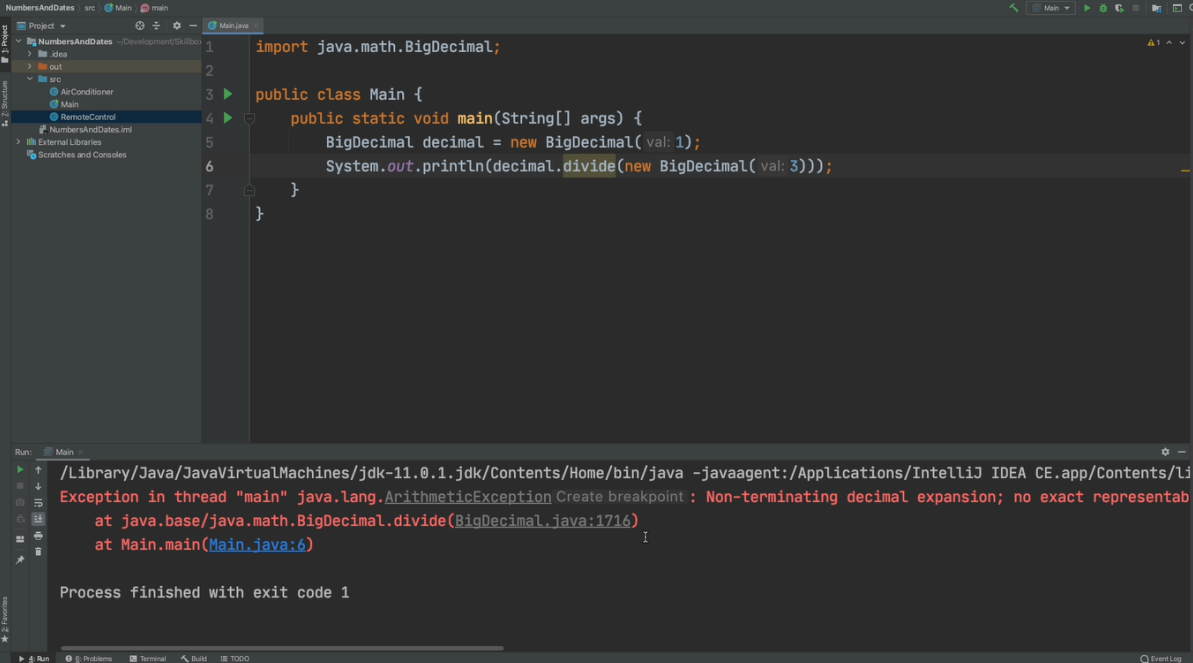
Process finished with exit code 0

У класса `BigDecimal` также есть различные методы, позволяющие производить с этими числами всевозможные математические операции. Наверняка вам интересны нестандартные случаи.



Например, если мы число 1 разделим на число 3 (число 1 можно здесь передать в явном виде, поскольку оно целое).

Запускаем и видим, что произошло исключение — Non-terminating decimal expansion; no exact representable decimal result; (невозможно в десятичной системе в точности представить результат выполнения этой операции).



```
import java.math.BigDecimal;

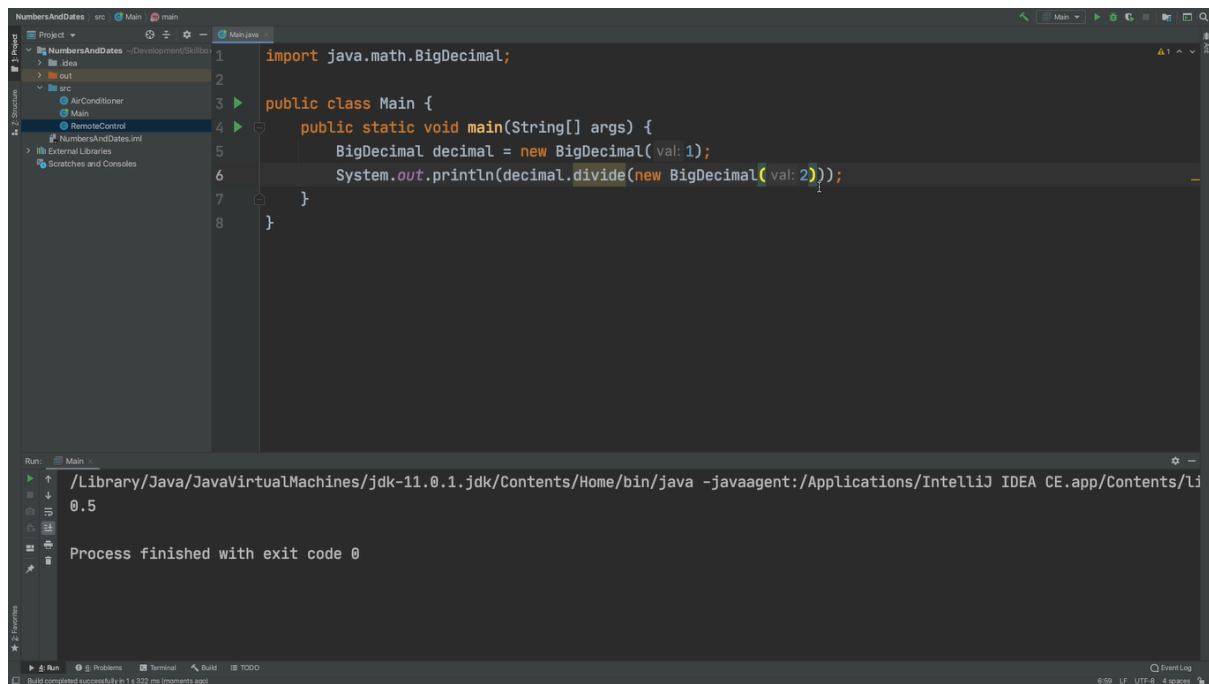
public class Main {
    public static void main(String[] args) {
        BigDecimal decimal = new BigDecimal(1);
        System.out.println(decimal.divide(new BigDecimal(3)));
    }
}
```

Run: Main

```
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib
Exception in thread "main" java.lang.ArithmeticException: Create breakpoint : Non-terminating decimal expansion; no exact representab
at java.base/java.math.BigDecimal.divide(BigDecimal.java:1716)
at Main.main(Main.java:6)

Process finished with exit code 1
```

Попробуем поделить на 2, чтобы убедиться, что вообще метод divide у класса BigDecimal работает. Видим, что всё в порядке. Но если поделить на 0, то тоже произойдёт исключение и будет написано division by zero, то есть деление на ноль.



The screenshot shows an IDE window with a project named 'NumbersAndDates'. The 'src' folder contains a 'Main' class. The code in 'Main.java' is as follows:

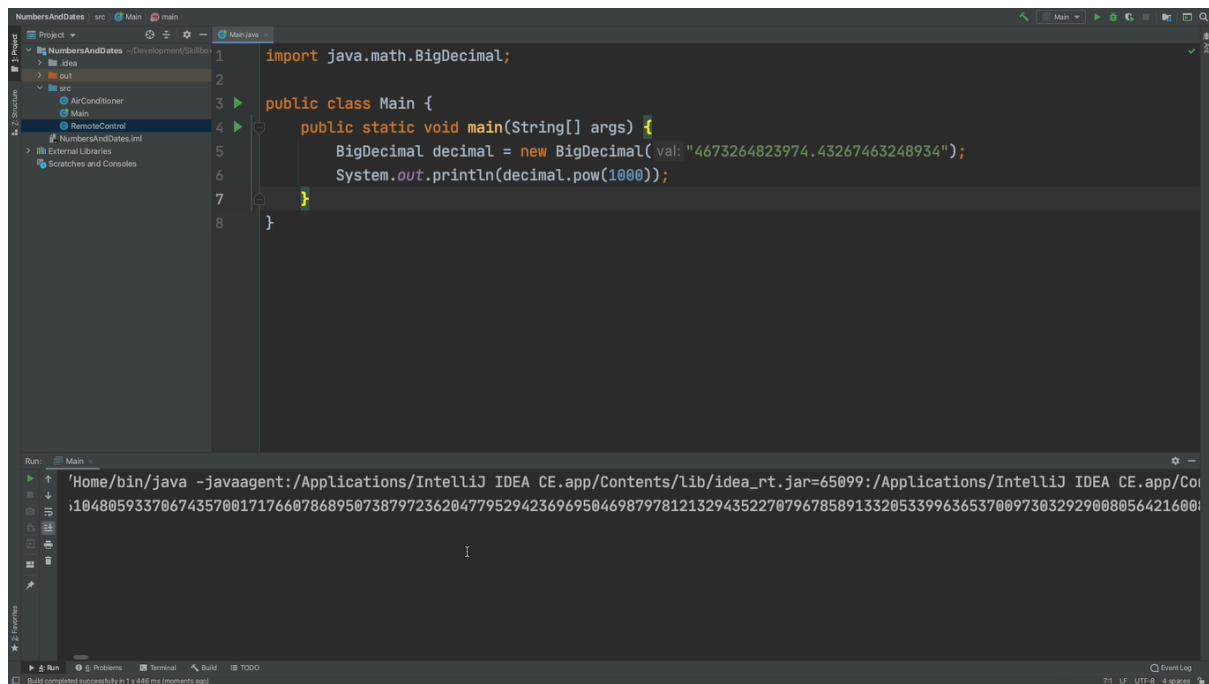
```
1 import java.math.BigDecimal;
2
3 public class Main {
4     public static void main(String[] args) {
5         BigDecimal decimal = new BigDecimal("1");
6         System.out.println(decimal.divide(new BigDecimal("2")));
7     }
8 }
```

The output of the program is displayed in the 'Run' console at the bottom:

```
Run: Main
/Library/Java/JavaVirtualMachines/jdk-11.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/Lib
0.5
Process finished with exit code 0
```

Кстати, некоторые числа, например, тот же самый 0, присутствуют в классе `BigDecimal` в качестве константы.

Есть множество других методов, позволяющих производить с числами различные математические операции. Главное, что работа с объектами класса `BigDecimal` может происходить без потери точности. Например, если мы возьмём какое-нибудь нецелое число и возведём это число в тысячную степень, то всё работает правильно:



**09:14-10:07**

## Выводы

В этой теме вы познакомились с классами `BigInteger` и `BigDecimal`, которые позволяют не только работать с очень большими числами, но и производить максимально точные вычисления.

Обратите внимание: для большинства повседневных задач такая высокая точность не требуется и классических типов `float` и `double` будет вполне достаточно. `BigInteger` и `BigDecimal` используют только тогда, когда это действительно необходимо: для больших и сложных финансовых или специальных научных расчётов.

На этом мы заканчиваем знакомство с числами. В следующей теме поговорим, как в Java можно работать с датой и временем, с классами, при помощи которых возможна такая работа, и с вариантами их использования.