Переменные, типы данных, операторы

Класы BigInteger и BigDecimal в Java

Встроенные примитивные числовые типы не всегда могут подходить для определенных программ. Например, необходимо хранить и использовать в программе очень большие числа, которые выходят за пределы допустимых значений для типов long и double.

Рассмотрим классы BigDecimal и BigInteger. Опишем два типа данных, их характеристики и сценарии их использования. Кратко рассмотрим различные операции с использованием этих двух классов.

BigDecimal представляет собой неизменяемое десятичное число со знаком произвольной точности. Он состоит из двух частей:

1.Возможность задать **размерность (масштаб, scale),** которая представляет собой количество цифр после десятичной точки
2.Возможность задать метод округления

Например, BigDecimal 3.14 имеет не масштабированное значение 314 и размерность 2.

BigDecimal необходим для высокоточной арифметики. Также его используют для вычислений, требующих контроля над размерностью и поведением округления.

BigDecimal в Java

Создать BigDecimal из (скалярного) числа типа double

```
BigDecimal bd = new BigDecimal( val: 1.0);

System.out.println("bd = " + bd);
```

```
bd = 1
```

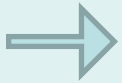
```
BigDecimal bd = new BigDecimal( val: 1.2);

System.out.println("bd = " + bd);
```

```
bd = 1.19999999999999955591079014993738383054733276367187 5
```

BigDecimal в Java

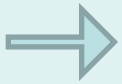
Лучше создание из строки:

```
BigDecimal bd = new BigDecimal( val: "1.0");

System.out.println("bd = " + bd);
```



```
bd = 1.0
```

```
BigDecimal bd = new BigDecimal( val: "1.2");

System.out.println("bd = " + bd);
```



```
bd = 1.2
```

BigDecimal в Java

Округление и масштабирование

Количество цифр после запятой (масштаб) - метод .setScale(scale). Часто одновременное указание вместе с масштабом режима округления с помощью .setScale(scale, roundingMode). Режим округления задаёт правило округления числа.

```
BigDecimal bd = new BigDecimal( val: 1.2);

bd.setScale(1); // получаем исключение ArithmeticException

System.out.println("bd = " + bd);
```

```
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : Rounding necessary
    at java.base/java.math.BigDecimal.commonNeedIncrement(BigDecimal.java:4628)
    at java.base/java.math.BigDecimal.needIncrement(BigDecimal.java:4835)
    at java.base/java.math.BigDecimal.divideAndRound(BigDecimal.java:4810)
    at java.base/java.math.BigDecimal.setScale(BigDecimal.java:2910)
    at java.base/java.math.BigDecimal.setScale(BigDecimal.java:2952)
    at com.solution.narozhnyy.MainBig.main(MainBig.java:10)
```

не известно, как округлить

BigDecimal в Java

Округление и масштабирование

Есть восемь вариантов режима округления:

ROUND_CEILING: В большую сторону
 0.333 -> 0.34
-0.333 -> -0.33

ROUND_DOWN: Отбрасывание разряда
 0.333 -> 0.33
-0.333 -> -0.33

ROUND_FLOOR: В меньшую сторону
 0.333 -> 0.33
-0.333 -> -0.34

ROUND_HALF_UP: Округление вверх, если число после запятой >= .5
0.5 -> 1.0
0.4 -> 0.0

ROUND_HALF_DOWN: Округление вверх, если число после запятой > .5
0.5 -> 0.0
0.6 -> 1.0

BigDecimal в Java

Округление и масштабирование

Есть восемь вариантов режима округления:

ROUND_HALF_EVEN:

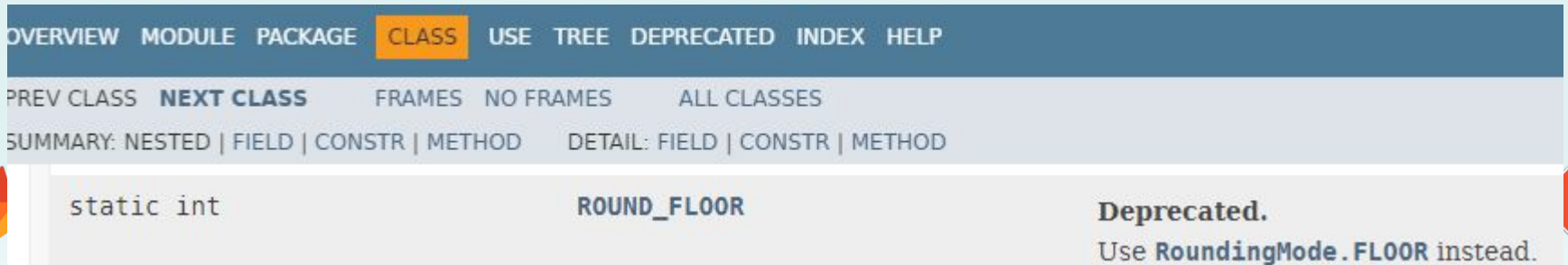
Округление половины по чётности округляет как обычно. Однако, когда округляемая цифра 5, округление будет идти вниз, если цифра слева от 5 чётная и вверх, если нечётная.

```
BigDecimal a = new BigDecimal( val: "2.5"); // цифра слева от 5 чётная, поэтому округление вниз

BigDecimal b = new BigDecimal( val: "1.5"); // цифра слева от 5 нечётная, поэтому округление вверх

System.out.println("a = " + a.setScale( newScale: 0, BigDecimal.ROUND_HALF_EVEN));

System.out.println("b = " + b.setScale( newScale: 0, BigDecimal.ROUND_HALF_EVEN));
```

```
a = 2
b = 2
```

@Deprecated(since = "9")

Согласно документам, setScale(int, int) не рекомендуется с Java 1.5. Окончательно устарело в Java 9



BigDecimal в Java

Округление и масштабирование. Enum RoundingMode

Задает поведение округления для числовых операций, допускающих потерю точности. Каждый режим округления указывает, как должна быть вычислена наименее значимая возвращаемая цифра округленного результата. Если возвращается меньше цифр, чем цифр, необходимых для представления точного числового результата, отброшенные цифры будут называться отброшенной дробью, независимо от вклада цифр в значение числа.

Описание каждого режима округления включает таблицу, в которой перечислено, как различные двузначные десятичные значения будут округляться до однозначного десятичного значения в рассматриваемом режиме округления. Столбец результатов в таблицах можно получить, создав число **BigDecimal** с указанным значением

BigDecimal в Java

Округление и масштабирование. Enum RoundingMode

Сводка операций округления при различных режимах округления

	Result of rounding input to one digit with the given rounding mode							
Input Number	UP	DOWN	CEILING	FLOOR	HALF_UP	HALF_DOWN	HALF_EVEN	UNNECESSARY
5.5	6	5	6	5	6	5	6	throw ArithmeticException
2.5	3	2	3	2	3	2	2	throw ArithmeticException
1.6	2	1	2	1	2	2	2	throw ArithmeticException
1.1	2	1	2	1	1	1	1	throw ArithmeticException
1.0	1	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-1	-2	-1	-1	-1	throw ArithmeticException
-1.6	-2	-1	-1	-2	-2	-2	-2	throw ArithmeticException
-2.5	-3	-2	-2	-3	-3	-2	-2	throw ArithmeticException
-5.5	-6	-5	-5	-6	-6	-5	-6	throw ArithmeticException

BigDecimal в Java

Округление и масштабирование. Enum RoundingMode

```
BigDecimal bd = new BigDecimal( val: 1.2);

System.out.println("bd = " + bd);

System.out.println("bd = " + bd.setScale( newScale: 1, RoundingMode.HALF_DOWN));

System.out.println("bd = " + bd.setScale( newScale: 1, BigDecimal.ROUND_HALF_DOWN));
```

```
bd = 1.19999999999999995559107901499373838305473327636718750
bd = 1.2
bd = 1.2
```

BigDecimal в Java

Неизменяемость и арифметика

Классы больших чисел не используют в своей работе операторы +-*/, а предоставляют вместо этого набор методов.

Числа BigDecimal являются неизменными. Это означает, что если создаётся новый объект BigDecimal со значением "2.00", такой объект останется "2.00" и никогда не может быть изменён.

Методы .add(), .multiply() и другие возвращают новый объект BigDecimal, содержащий результат.

Поэтому

```
BigDecimal bd = new BigDecimal( val: 1.2);
BigDecimal bd1 = new BigDecimal( val: 1.4);

bd = bd.add( bd1 );
//bd.add( bd1 );

System.out.println("bd = " + bd);
System.out.println("bd = " + bd.setScale( newScale: 1, RoundingMode.HALF_DOWN));
System.out.println("bd = " + bd.setScale( newScale: 1, BigDecimal.ROUND_HALF_DOWN));
```

```
bd = 2.599999999999999866773237044981215149164199829101562 5
bd = 2.6
bd = 2.6
```

BigDecimal в Java

Неизменяемость и арифметика

А так неверно

```
BigDecimal bd = new BigDecimal( val: 1.2);
BigDecimal bd1 = new BigDecimal( val: 1.4);

//bd = bd.add( bd1 );
bd.add( bd1 );

System.out.println("bd = " + bd);
System.out.println("bd = " + bd.setScale( newScale: 1, RoundingMode.HALF_DOWN));
System.out.println("bd = " + bd.setScale( newScale: 1, BigDecimal.ROUND_HALF_DOWN));
```

```
bd = 1.19999999999999995559107901499373838305473327636718 75
bd = 1.2
bd = 1.2
```

BigDecimal в Java

Сравнение

Важно! Никогда не использовать для сравнения объектов **BigDecimal** метод **.equals()**. Этого нельзя делать потому, что функция **equals** будет сравнивать размерности и значения. Если размерности различаются, **.equals()** вернёт ложь, даже если они математически равны:

```
BigDecimal bd = new BigDecimal( val: "1.20");
BigDecimal bd1 = new BigDecimal( val: "1.2");

System.out.println("bd.equals(bd1) = " + bd.equals(bd1));
```

```
bd.equals(bd1) = false
```

Следует использовать методы **.compareTo()** и **.signum()**.

```
BigDecimal bd = new BigDecimal( val: "1.20");
BigDecimal bd1 = new BigDecimal( val: "1.2");
 // возвращает (-1 если a < b), (0 если a == b), (1 если a > b)
System.out.println("bd.compareTo(bd1) = " + bd.compareTo(bd1));
// возвращает (-1 если a < 0), (0 если a == 0), (1 если a > 0)
System.out.println("bd.signum() = " + bd.signum());
```

```
bd.compareTo(bd1) = 0
bd.signum() = 1
```

BigDecimal в Java

Основные методы класса BigDecimal:

- BigDecimal add(BigDecimal other): возвращает сумму двух чисел
- BigDecimal subtract(BigDecimal other): возвращает разность двух чисел
- BigDecimal multiply(BigDecimal other): возвращает произведение двух чисел
- BigDecimal divide(BigDecimal other): возвращает частное двух чисел
- BigDecimal divide(BigDecimal other, RoundingMode mode): результат деления двух чисел, округленное в соответствии с режимом mode
- int compareTo(BigDecimal other): сравнивает два числа. Возвращает -1, если текущий объект меньше числа other, 1 - если текущий объект больше и 0 - если числа равны
- static BigDecimal valueOf(double x): возвращает объект BigDecimal, значение которого равно числу, переданному в качестве параметра
- double doubleValue(): преобразует объект BigDecimal в double
- float floatValue(): преобразует объект BigDecimal в float

BigInteger в Java

Основные методы класса BigInteger:

- BigInteger add(BigInteger other): возвращает сумму двух чисел
- BigInteger subtract(BigInteger other): возвращает разность двух чисел
- BigInteger multiply(BigInteger other): возвращает произведение двух чисел
- BigInteger divide(BigInteger other): возвращает частное двух чисел
- BigInteger mod(BigInteger other): возвращает остаток от целочисленного деления двух чисел
- BigInteger sqrt(): возвращает квадратный корень числа
- int compareTo(BigInteger other): сравнивает два числа. Возвращает -1, если текущий объект меньше числа other, 1 - если текущий объект больше и 0 - если числа равны
- static BigInteger valueOf(long x): возвращает объект BigInteger, значение которого равно числу, переданному в качестве параметра
- int intValue(): конвертирует объект BigInteger в объект int
- byte byteValue(): преобразует объект BigInteger в byte
- short shortValue(): преобразует объект BigInteger в short
- long longValue(): преобразует объект BigInteger в long

BigInteger в Java

```
BigInteger bi0 = new BigInteger( val: "1111111111111111111111111111111111111111");
BigInteger bi1 = new BigInteger( val: "1111111111111111111111111111111111111111");

bi0 = bi0.add( bi1 );
```

```
bi0 = 2222222222222222222222222222222222222222
```

Использование строки для передачи нужного числа — хороший вариант. Оба класса умеют автоматически извлекать из переданных строк числовые значения.

Передача строки в качестве параметра — только один из возможных конструкторов.

Числа превышают максимальные значения long и double.

Просто передать в конструктор число

1111111111111111111111111111111111111111

не получится: Java попытается «вместить» переданное число в один из примитивных типов данных, но оно не влезет.

Как и **BigDecimal в** классе **BigInteger** объекты являются неизменными (**Immutable**).

Спасибо за внимание!