IX. Типы для времени и денег

3. BigDecimal

Зачем нужен BigDecimal?

```
public class BigDecimalDemo {

    public static void main(String[] args) {

        double A = 2.0;
        double B = 1.1;
        double C = A - B;

        BigDecimal bigA = new BigDecimal("2.0");
        BigDecimal bigB = new BigDecimal("1.1");
        BigDecimal bigC = bigA.subtract(bigB);

        System.out.println("Using double " + A + " - " + B + " = " + C);
        System.out.println("Using BigDecimal " + bigA + " - " + bigB + " = " + bigC);
    }
}
```

```
Using double 2.0 - 1.1 = 0.8999999999999999
Using BigDecimal 2.0 - 1.1 = 0.9
```

BigDecimal позволяет точно представлять числа записанные в десятичной системе и точно выполнять арифметические операции.

Класс BigDecimal

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    private volatile BigInteger intVal;
    private int scale;

    private transient long intCompact;
    final static long INFLATED = Long.MIN_VALUE;

    public int scale() {
        return scale;
    }

    public BigDecimal setScale(int newScale)

    public BigInteger unscaledValue() {
        return this.inflate();
    }

    private BigInteger inflate() {
        if (intVal == null)
            intVal = BigInteger.valueOf(intCompact);
        return intVal;
    }
}
```

Класс BigDecimal позволяет представлять десятичные числа с произвольной точностью. Объекты класса BigDecimal являются неизменными. BigDecimal включает мантиссу в виде BigInteger и десятичный порядок в виде int. Десятичное число представляемое BigDecimal равно мантиссе поделённой на 10 в степени десятичного порядка. (мантисса × 10^-порядок). Если значение мантиссы попадает в диапазон long за исключением нижней границы может использоваться компактное представление мантиссы с помощью long. С помощью методов unscaledValue и scale можно получить мантиссу и десятичный порядок. С помощью метода precision() можно получить количество разрядов в мантиссе называемое разрядностью.

Класс BigDecimal

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    private transient int precision;

    public int precision() {
        int result = precision;
        if (result == 0) {
            long s = intCompact;
            if (s != INFLATED)
                result = longDigitLength(s);
            else
                result = bigDigitLength(inflate());
            precision = result;
        }
        return result;
    }
}
```

Хранение BigDecimal

$$unscaledValue \times 10^{(-scale)}$$

$2.00 = 200 \times 10^{-2}$
unscaledValue: 200
scale: 2
precision: 3

$2.0 = 20 \times 10^{-1}$
unscaledValue: 20
scale: 1
precision: 2

Существует много представлений для одного числа с помощью BigDecimal.

Десятичный порядок, мантисса и разрядность

```
public class ScaleValuePrecisionDemo {

    public static void main(String[] args) {

        BigDecimal big = new BigDecimal("123.456789");

        System.out.println("Big decimal: " + big);
        System.out.println("Value: " + big.unscaledValue());
        System.out.println("Scale: " + big.scale());
        System.out.println("Precision: " + big.precision());
    }
}
```

```
Big decimal: 123.456789
Value: 123456789
Scale: 6
Precision: 9
```

“Изменение” десятичного порядка

```
public class SetScaleDemo {

    public static void main(String[] args) {

        BigDecimal big = new BigDecimal("200");

        System.out.println("Big decimal: " + big);
        System.out.println("Unscaled value: " + big.unscaledValue());
        System.out.println("Scale: " + big.scale());
        System.out.println("Precision: " + big.precision());

        big = big.setScale(2);

        System.out.println("Big decimal: " + big);
        System.out.println("Unscaled value: " + big.unscaledValue());
        System.out.println("Scale: " + big.scale());
        System.out.println("Precision: " + big.precision());

        big = big.setScale(-2);

        System.out.println("Big decimal: " + big);
        System.out.println("Unscaled value: " + big.unscaledValue());
        System.out.println("Scale: " + big.scale());
        System.out.println("Precision: " + big.precision());
    }
}
```

“Изменение” десятичного порядка

```
Big decimal: 200
Unscaled value: 200
Scale: 0
Precision: 3

Big decimal: 200.00
Unscaled value: 20000
Scale: 2
Precision: 5

Big decimal: 2E+2
Unscaled value: 2
Scale: -2
Precision: 1
```

Неизменность BigDecimal

```
public class ImmutableDemo {

    public static void main(String[] args) {

        BigDecimal initial = new BigDecimal("0");

        BigDecimal term1 = new BigDecimal("0.001");
        BigDecimal term2 = new BigDecimal("0.002");
        BigDecimal term3 = new BigDecimal("0.003");
        BigDecimal term4 = new BigDecimal("0.004");
        BigDecimal term5 = new BigDecimal("0.005");

        BigDecimal result = initial.add(term1).add(term2).add(term3).add(term4).add(term5);

        System.out.println("Equal references ? " + (initial == result));

        System.out.println("Initial value " + initial);
        System.out.println("Result value  " + result);
    }
}
```

```
Equal references ? false
Initial value 0
Result value  0.015
```

Проверка на равенство

Проверка на равенство

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public boolean equals(Object x) {
        if (!(x instanceof BigDecimal))
            return false;
        BigDecimal xDec = (BigDecimal) x;
        if (x == this)
            return true;
        if (scale != xDec.scale )
            return false;
        long s = this.intCompact;
        long xs = xDec.intCompact;
        if (s != INFLATED) {
            if (xs == INFLATED)
                xs = compactValFor(xDec.intVal);
            return xs == s;
        } else if (xs != INFLATED)
            return xs == compactValFor(this.intVal);

        return this.inflate().equals(xDec.inflate());
    }
}
```

Метод equals() сравнивает десятичный порядок и мантиссу по отдельности поэтому различные представления одного числа не равны.

Проверка на равенство

```
public class EqualsDemo {

    public static void main(String[] args) {

        BigDecimal bigA = new BigDecimal("2.00");
        BigDecimal bigB = new BigDecimal("2.0");

        System.out.println("Big decimal A: " + bigA + " unscaled value: " + bigA.unscaledValue() +
                " scale: " + bigA.scale() + " precision: " + bigA.precision());

        System.out.println("Big decimal B: " + bigB + " unscaled value: " + bigB.unscaledValue() +
                " scale: " + bigB.scale() + " precision: " + bigB.precision());

        System.out.println("Are A and B equal ? " + bigA.equals(bigB));

        bigA = bigA.stripTrailingZeros();
        System.out.println("Big decimal A: " + bigA + " unscaled value: " + bigA.unscaledValue() +
                " scale: " + bigA.scale() + " precision: " + bigA.precision());

        bigB = bigB.stripTrailingZeros();
        System.out.println("Big decimal B: " + bigB + " unscaled value: " + bigB.unscaledValue() +
                " scale: " + bigB.scale() + " precision: " + bigB.precision());

        System.out.println("Are A and B equal ? " + bigA.equals(bigB));
        System.out.println("Equal references? " + (bigA == bigB));
    }
}
```

```
Big decimal A: 2.00 unscaled value: 200 scale: 2 precision: 3
Big decimal B: 2.0 unscaled value: 20 scale: 1 precision: 2
Are A and B equal ? false
Big decimal A: 2 unscaled value: 2 scale: 0 precision: 1
Big decimal B: 2 unscaled value: 2 scale: 0 precision: 1
Are A and B equal ? true
Equal references? false
```

Использование проверки на равенство

```
public class HashSetDemo {

    public static void main(String[] args) {

        Set<BigDecimal> set = new HashSet<BigDecimal>();

        set.add(new BigDecimal("2"));
        set.add(new BigDecimal("2.0"));
        set.add(new BigDecimal("2.00"));
        set.add(new BigDecimal("2.000"));

        System.out.println("HashSet size: " + set.size());
        System.out.println("HashSet contents: " + set);
    }
}
```

```
HashSet size: 4
HashSet contents: [2.00, 2.000, 2, 2.0]
```

Сравнение

Сравнение

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public int compareTo(BigDecimal val) {

        if (scale == val.scale) {
            long xs = intCompact;
            long ys = val.intCompact;
            if (xs != INFLATED && ys != INFLATED)
                return xs != ys ? ((xs > ys) ? 1 : -1) : 0;
        }
        int xsign = this.signum();
        int ysign = val.signum();
        if (xsign != ysign)
            return (xsign > ysign) ? 1 : -1;
        if (xsign == 0)
            return 0;
        int cmp = compareMagnitude(val);
        return (xsign > 0) ? cmp : -cmp;
    }

    private int compareMagnitude(BigDecimal val) {
        ...
    }

    public BigDecimal min(BigDecimal val) {
        return (compareTo(val) <= 0 ? this : val);
    }

    public BigDecimal max(BigDecimal val) {
        return (compareTo(val) >= 0 ? this : val);
    }
}
```

В классе BigDecimal естественный порядок несовместим с определением метода equals. С точки зрения метода compareTo различные представления одного числа равны.

Сравнение

```
public class CompareDemo {

    public static void main(String[] args) {

        BigDecimal bigA = new BigDecimal("2.00");
        BigDecimal bigB = new BigDecimal("2.0");

        System.out.println("Big decimal A: " + bigA + " unscaled value: " + bigA.unscaledValue() +
                " scale: " + bigA.scale() + " precision: " + bigA.precision());

        System.out.println("Big decimal B: " + bigB + " unscaled value: " + bigB.unscaledValue() +
                " scale: " + bigB.scale() + " precision: " + bigB.precision());

        System.out.println("Are A and B equal ? " + (bigA.compareTo(bigB) == 0));
    }
}
```

```
Big decimal A: 2.00 unscaled value: 200 scale: 2 precision: 3
Big decimal B: 2.0 unscaled value: 20 scale: 1 precision: 2
Are A and B equal ? true
```

Использование сравнения

```
public class TreeSetDemo {

    public static void main(String[] args) {

        Set<BigDecimal> set = new TreeSet<BigDecimal>();

        set.add(new BigDecimal("2"));
        set.add(new BigDecimal("2.0"));
        set.add(new BigDecimal("2.00"));
        set.add(new BigDecimal("2.000"));

        System.out.println("TreeSet size: " + set.size());
        System.out.println("TreeSet contents: " + set);
    }
}
```

```
TreeSet size: 1
TreeSet contents: [2]
```

Использование сравнения

```
public class TreeSetCompareDemo {

    public static void main(String[] args) {

        Set<BigDecimal> set = new TreeSet<BigDecimal>();

        set.add(new BigDecimal("2"));
        set.add(new BigDecimal("3.0"));
        set.add(new BigDecimal("4.00"));
        set.add(new BigDecimal("5.000"));
        set.add(new BigDecimal("8.00"));
        set.add(new BigDecimal("7.0"));
        set.add(new BigDecimal("6.00"));
        set.add(new BigDecimal("-1.0"));
        set.add(new BigDecimal("-2.00"));
        set.add(new BigDecimal("-3.0"));
        set.add(new BigDecimal("0.000"));

        System.out.println("TreeSet size: " + set.size());
        System.out.println("TreeSet contents: " + set);
    }
}
```

```
TreeSet size: 11
TreeSet contents: [-3.0, -2.00, -1.0, 0.000, 2, 3.0, 4.00, 5.000, 6.00, 7.0, 8.00]
```

Получение BigDecimal

Конструкторы BigDecimal

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public BigDecimal(char[] in, int offset, int len)
    public BigDecimal(char[] in)
    public BigDecimal(String val)
    public BigDecimal(double val)
    public BigDecimal(int val)
    public BigDecimal(long val)
    public BigDecimal(BigInteger val)
    public BigDecimal(BigInteger unscaledVal, int scale)

    BigDecimal(BigInteger intVal, long val, int scale, int prec) {
        this.scale = scale;
        this.precision = prec;
        this.intCompact = val;
        this.intVal = intVal;
    }
}
```

Конструктор принимающий double в качестве параметра использовать не следует из-за неточного представления double.

Конструктор принимающий double

```
public class NoDoubleConstructorDemo {

    public static void main(String[] args) {

        System.out.println(new BigDecimal(0.5));
        System.out.println(new BigDecimal(0.25));
        System.out.println(new BigDecimal(0.125));
        System.out.println(new BigDecimal(0.0625));
        System.out.println(new BigDecimal(0.03125));

        System.out.println(new BigDecimal(0.1));
        System.out.println(new BigDecimal(0.01));
        System.out.println(new BigDecimal(0.001));
        System.out.println(new BigDecimal(0.0001));
        System.out.println(new BigDecimal(0.00001));
    }
}
```

```
0.5
0.25
0.125
0.0625
0.03125
0.1000000000000000055511151231257827021181583404541015625
0.0100000000000000002081668171172168513294309377670288085937 5
0.0010000000000000000208166817117216851329430937767028808593 75
0.0001000000000000000047921736023859295983129413798451423645019531 25
0.0000100000000000000000818030539140313095458623138256371021270751953125
```

Конструктор принимающий String

```
public class YesStringConstructorDemo {

    public static void main(String[] args) {

            System.out.println(new BigDecimal("0.5"));
            System.out.println(new BigDecimal("0.25"));
            System.out.println(new BigDecimal("0.125"));
            System.out.println(new BigDecimal("0.0625"));
            System.out.println(new BigDecimal("0.03125"));

            System.out.println(new BigDecimal("0.1"));
            System.out.println(new BigDecimal("0.01"));
            System.out.println(new BigDecimal("0.001"));
            System.out.println(new BigDecimal("0.0001"));
            System.out.println(new BigDecimal("0.00001"));
     }
}
```

```
0.5
0.25
0.125
0.0625
0.03125
0.1
0.01
0.001
0.0001
0.00001
```

Кеш экземпляров BigDecimal

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    private static final BigDecimal zeroThroughTen[] = {
        new BigDecimal(BigInteger.ZERO,     0,  0, 1),
        new BigDecimal(BigInteger.ONE,      1,  0, 1),
        new BigDecimal(BigInteger.valueOf(2),   2,  0, 1),
        new BigDecimal(BigInteger.valueOf(3),   3,  0, 1),
        new BigDecimal(BigInteger.valueOf(4),   4,  0, 1),
        new BigDecimal(BigInteger.valueOf(5),   5,  0, 1),
        new BigDecimal(BigInteger.valueOf(6),   6,  0, 1),
        new BigDecimal(BigInteger.valueOf(7),   7,  0, 1),
        new BigDecimal(BigInteger.valueOf(8),   8,  0, 1),
        new BigDecimal(BigInteger.valueOf(9),   9,  0, 1),
        new BigDecimal(BigInteger.TEN,      10, 0, 2),
    };

    private static final BigDecimal[] ZERO_SCALED_BY = {
        zeroThroughTen[0],
        new BigDecimal(BigInteger.ZERO, 0, 1, 1),
        new BigDecimal(BigInteger.ZERO, 0, 2, 1),
        new BigDecimal(BigInteger.ZERO, 0, 3, 1),
        new BigDecimal(BigInteger.ZERO, 0, 4, 1),
        new BigDecimal(BigInteger.ZERO, 0, 5, 1),
        new BigDecimal(BigInteger.ZERO, 0, 6, 1),
        new BigDecimal(BigInteger.ZERO, 0, 7, 1),
        new BigDecimal(BigInteger.ZERO, 0, 8, 1),
        new BigDecimal(BigInteger.ZERO, 0, 9, 1),
        new BigDecimal(BigInteger.ZERO, 0, 10, 1),
        new BigDecimal(BigInteger.ZERO, 0, 11, 1),
        new BigDecimal(BigInteger.ZERO, 0, 12, 1),
        new BigDecimal(BigInteger.ZERO, 0, 13, 1),
        new BigDecimal(BigInteger.ZERO, 0, 14, 1),
        new BigDecimal(BigInteger.ZERO, 0, 15, 1),
    };
}
```

Статические фабрики BigDecimal

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public static BigDecimal valueOf(long unscaledVal, int scale) {
        if (scale == 0)
            return valueOf(unscaledVal);
        else if (unscaledVal == 0) {
            if (scale > 0 && scale < ZERO_SCALED_BY.length)
                return ZERO_SCALED_BY[scale];
            else
                return new BigDecimal(BigInteger.ZERO, 0, scale, 1);
        }
        return new BigDecimal(unscaledVal == INFLATED ?
                              BigInteger.valueOf(unscaledVal) : null,
                              unscaledVal, scale, 0);
    }

    public static BigDecimal valueOf(long val) {
        if (val >= 0 && val < zeroThroughTen.length)
            return zeroThroughTen[(int)val];
        else if (val != INFLATED)
            return new BigDecimal(null, val, 0, 0);
        return new BigDecimal(BigInteger.valueOf(val), val, 0, 0);
    }

    public static BigDecimal valueOf(double val) {
        return new BigDecimal(Double.toString(val));
    }
}
```

Кеширование целых чисел 0 - 10

```
public class CachingDemo {

    public static void main(String[] args) {

        BigDecimal i = new BigDecimal(10);
        BigDecimal j = BigDecimal.valueOf(10);

        System.out.println("i = " + i + ", j = " + j + ", i==j is " + (i == j));

        BigDecimal k = BigDecimal.valueOf(10);
        BigDecimal l = BigDecimal.valueOf(10);

        System.out.println("k = " + k + ", l = " + l + ", k==l is " + (k == l));

        BigDecimal m = BigDecimal.valueOf(11);
        BigDecimal n = BigDecimal.valueOf(11);

        System.out.println("m = " + m + ", n = " + n + ", m==n is " + (m == n));
    }
}
```

```
i = 10, j = 10, i==j is false
k = 10, l = 10, k==l is true
m = 11, n = 11, m==n is false
```

Кеширование нулей

```
public class CachingZeroDemo {

    public static void main(String[] args) {

        BigDecimal i = new BigDecimal(BigInteger.ZERO, 15);
        BigDecimal j = BigDecimal.valueOf(0, 15);

        System.out.println("i = " + i + ", j = " + j + ", i==j is " + (i == j));

        BigDecimal k = BigDecimal.valueOf(0, 15);
        BigDecimal l = BigDecimal.valueOf(0, 15);

        System.out.println("k = " + k + ", l = " + l + ", k==l is " + (k == l));

        BigDecimal m = BigDecimal.valueOf(0, 16);
        BigDecimal n = BigDecimal.valueOf(0, 16);

        System.out.println("m = " + m + ", n = " + n + ", m==n is " + (m == n));
    }
}
```

```
i = 0E-15, j = 0E-15, i==j is false
k = 0E-15, l = 0E-15, k==l is true
m = 0E-16, n = 0E-16, m==n is false
```

Точные арифметические операции

Точные арифметические операции

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public BigDecimal add(BigDecimal augend)
    public BigDecimal subtract(BigDecimal subtrahend)
    public BigDecimal multiply(BigDecimal multiplicand)
    public BigDecimal divide(BigDecimal divisor)
    public BigDecimal setScale(int newScale)

}
```

	Методы add, subtract и multiply всегда возвращают точный результат. Десятичный порядок результата равен так называемому предпочтительному порядку. Метод divide может выбросить исключение ArithmeticException если точный результат не может быть представлен используя конечное число разрядов, например при делении 1 на 3. Десятичный порядок результата может быть больше предпочтительного порядка если это необходимо, например при делении 1 на 32. Метод setScale возвращает BigDecimal с заданным десятичным порядком и с тем же численным значением что и исходный BigDecimal. Этот метод выбрасывает исключение ArithmeticException если операция изменения десятичного порядка требует округления.

Предпочтительный порядок

Операция	Порядок	Порядок	Предпочтительный порядок
A.add(B)	A.scale()	B.scale()	Max(A.scale(),B.scale())
A.subtract(B)	A.scale()	B.scale()	Max(A.scale(),B.scale())
A.multiply(B)	A.scale()	B.scale()	A.scale()+B.scale()
A.divide(B)	A.scale()	B.scale()	A.scale()-B.scale()

Точные арифметические операции

```
public class PreciseArithmeticDemo {

    public static void main(String[] args) {

        BigDecimal a = new BigDecimal("0.002");
        BigDecimal b = new BigDecimal("0.001");

        System.out.println(a + " + " + b + " = " + a.add(b));
        System.out.println(a + " - " + b + " = " + a.subtract(b));
        System.out.println(a + " * " + b + " = " + a.multiply(b));
        System.out.println(a + " / " + b + " = " + a.divide(b));

        System.out.println();

        a = new BigDecimal("100");
        b = new BigDecimal("0.01");

        System.out.println(a + " + " + b + " = " + a.add(b) );
        System.out.println(a + " - " + b + " = " + a.subtract(b));
        System.out.println(a + " * " + b + " = " + a.multiply(b));
        System.out.println(a + " / " + b + " = " + a.divide(b));
    }
}
```

```
0.002 + 0.001 = 0.003
0.002 - 0.001 = 0.001
0.002 * 0.001 = 0.000002
0.002 / 0.001 = 2

100 + 0.01 = 100.01
100 - 0.01 = 99.99
100 * 0.01 = 1.00
100 / 0.01 = 1.00E+4
```

Точное деление

```
public class PreciseDivisionDemo {

    public static void main(String[] args) {

        BigDecimal a = new BigDecimal("1");
        BigDecimal b = new BigDecimal("32");
        System.out.println(a + " / " + b + " = " + a.divide(b));

        System.out.println();

        a = new BigDecimal("1");
        b = new BigDecimal("3");

        System.out.println(a + " / " + b + " = " + a.divide(b));
     }
}
```

```
1 / 32 = 0.03125

Exception in thread "main" java.lang.ArithmeticException: Non-terminating decimal expansion; no exact
representable decimal result.
at java.math.BigDecimal.divide(Unknown Source)
at bigdecimal.PreciseDivisionDemo.main(PreciseDivisionDemo.java:19)
```

Точное “изменение” десятичного порядка

```
public class ScaleDemo {

    public static void main(String[] args) {

        BigDecimal a = new BigDecimal("2.10000");

        System.out.println(a);
        System.out.println(a.setScale(4));
        System.out.println(a.setScale(3));
        System.out.println(a.setScale(2));
        System.out.println(a.setScale(1));
        System.out.println(a.setScale(0));
    }
}
```

```
2.10000
2.1000
2.100
2.10
2.1
Exception in thread "main" java.lang.ArithmeticException: Rounding necessary
at java.math.BigDecimal.divideAndRound(Unknown Source)
at java.math.BigDecimal.setScale(Unknown Source)
at java.math.BigDecimal.setScale(Unknown Source)
at bigdecimal.ScaleDemo.main(ScaleDemo.java:17)
```

Арифметические операции с контекстом

Класс MathContext

```
public final class MathContext implements Serializable {

    final int precision;
    final RoundingMode roundingMode;

    public MathContext(int setPrecision) {
        this(setPrecision, DEFAULT_ROUNDINGMODE);
        return;
    }

    public MathContext(int setPrecision, RoundingMode setRoundingMode) {

        if (setPrecision < MIN_DIGITS)
            throw new IllegalArgumentException("Digits < 0");
        if (setRoundingMode == null)
            throw new NullPointerException("null RoundingMode");

         precision = setPrecision;
         roundingMode = setRoundingMode;
         return;
    }

    public int getPrecision() {
        return precision;
    }

    public RoundingMode getRoundingMode() {
        return roundingMode;
    }
}
```

Класс MathContext позволяет задавать разрядность результата и способ округления. Для задания способа округления используется перечисление RoundingMode.

Перечисление RoundingMode

```
public enum RoundingMode {

    UP(BigDecimal.ROUND_UP),
    DOWN(BigDecimal.ROUND_DOWN),
    CEILING(BigDecimal.ROUND_CEILING),
    FLOOR(BigDecimal.ROUND_FLOOR),
    HALF_UP(BigDecimal.ROUND_HALF_UP),
    HALF_DOWN(BigDecimal.ROUND_HALF_DOWN),
    HALF_EVEN(BigDecimal.ROUND_HALF_EVEN),
    UNNECESSARY(BigDecimal.ROUND_UNNECESSARY);

    final int oldMode;

    private RoundingMode(int oldMode) {
        this.oldMode = oldMode;
    }
}
```

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public final static int ROUND_UP =          0;
    public final static int ROUND_DOWN =        1;
    public final static int ROUND_CEILING =     2;
    public final static int ROUND_FLOOR =       3;
    public final static int ROUND_HALF_UP =     4;
    public final static int ROUND_HALF_DOWN =   5;
    public final static int ROUND_HALF_EVEN =   6;
    public final static int ROUND_UNNECESSARY = 7;

    public BigDecimal divide(BigDecimal divisor, int roundingMode)
}
```

Перечисление RoundingMode задаёт способ округления арифметических операций с контекстом. Позволяет обходиться без статических констант из класса BigDecimal.

Перечисление RoundingMode

Способ округления	Описание
`UP`	округление от нуля, к большему по модулю целому значению
`DOWN`	округление к нулю, к меньшему по модулю целому значению
`CEILING`	округление в сторону большего целого
`FLOOR`	округление к меньшему целому
`HALF_UP`	округление к ближайшему целому, среднее значение округляется к большему целому
`HALF_DOWN`	округление к ближайшему целому, среднее значение округляется к меньшему целому
`HALF_EVEN`	округление к ближайшему целому, среднее значение округляется к четному числу
`UNNECESSARY`	предполагается, что результат будет целым, и округление не понадобится в противном случае будет выброшено исключение

Перечисление RoundingMode

	Результат округления используя различные способы округления							
Число	**UP**	**DOWN**	**CEILING**	**FLOOR**	**HALF_UP**	**HALF_DOWN**	**HALF_EVEN**	**UNNECESSARY**
5.5	6	5	6	5	6	5	6	ArithmeticException
2.5	3	2	3	2	3	2	2	ArithmeticException
1.6	2	1	2	1	2	2	2	ArithmeticException
1.1	2	1	2	1	1	1	1	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	ArithmeticException
-1.6	-2	-1	-1	-2	-2	-2	-2	ArithmeticException
-2.5	-3	-2	-2	-3	-3	-2	-2	ArithmeticException
-5.5	-6	-5	-5	-6	-6	-5	-6	ArithmeticException

Если используется способ округления UNNECESSARY, а округление необходимо будет выброшено исключение ArithmeticException.

Арифметические операции с контекстом

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public BigDecimal add(BigDecimal augend, MathContext mc)
    public BigDecimal subtract(BigDecimal subtrahend, MathContext mc)
    public BigDecimal multiply(BigDecimal multiplicand, MathContext mc)
    public BigDecimal divide(BigDecimal divisor, MathContext mc)

    public BigDecimal round(MathContext mc)
}
```

Для всех арифметических операций, результат будет таким, как будто сначала был получен точный промежуточный результат, а потом он был округлён до указанного числа разрядов заданных разрядностью (если необходимо), используя заданный способ округления. Разрядность и способ округления задаются с помощью MathContext.

Арифметические операции с контекстом

```
public class ContextArithmeticDemo {

    public static void main(String[] args) {

        BigDecimal a = new BigDecimal("2.00000");
        BigDecimal b = new BigDecimal("3.0");

        for (int i = 1; i < 10; i++) {
            MathContext context = new MathContext(i, RoundingMode.HALF_UP);
            System.out.println(a + " / " + b + " = " + a.divide(b, context));
        }
    }
}
```

```
2.00000 / 3.0 = 0.7
2.00000 / 3.0 = 0.67
2.00000 / 3.0 = 0.667
2.00000 / 3.0 = 0.6667
2.00000 / 3.0 = 0.66667
2.00000 / 3.0 = 0.666667
2.00000 / 3.0 = 0.6666667
2.00000 / 3.0 = 0.66666667
2.00000 / 3.0 = 0.666666667
```

“Старые” арифметические операции

“Старые” арифметические операции

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public final static int ROUND_UP =           0;
    public final static int ROUND_DOWN =         1;
    public final static int ROUND_CEILING =      2;
    public final static int ROUND_FLOOR =        3;
    public final static int ROUND_HALF_UP =      4;
    public final static int ROUND_HALF_DOWN =    5;
    public final static int ROUND_HALF_EVEN =    6;
    public final static int ROUND_UNNECESSARY =  7;

    public BigDecimal divide(BigDecimal divisor, int scale, int roundingMode)
    public BigDecimal setScale(int newScale, int roundingMode)

}
```

До появления перечислений в J2SE 5.0 для задания способа округления использовался набор статических целочисленных констант. В классе BigDecimal есть перегруженные варианты методов использующие эти константы. Кроме способа округления задаётся десятичный порядок.

“Старые” арифметические операции

```
public class OldDivideDemo {

    public static void main(String[] args) {

        BigDecimal a = new BigDecimal("2.00000");
        BigDecimal b = new BigDecimal("3.0");

        for (int i = 1; i < 10; i++) {
            System.out.println(a + " / " + b + " = "
                    + a.divide(b, i, BigDecimal.ROUND_HALF_UP));
        }

        for (int i = 1; i < 10; i++) {
            System.out.println(a + " / " + b + " = "
                    + a.divide(b, i, BigDecimal.ROUND_UNNECESSARY));
        }

    }
}
```

```
2.00000 / 3.0 = 0.7
2.00000 / 3.0 = 0.67
2.00000 / 3.0 = 0.667
2.00000 / 3.0 = 0.6667
2.00000 / 3.0 = 0.66667
2.00000 / 3.0 = 0.666667
2.00000 / 3.0 = 0.6666667
2.00000 / 3.0 = 0.66666667
2.00000 / 3.0 = 0.666666667
Exception in thread "main" java.lang.ArithmeticException: Rounding necessary
at java.math.BigDecimal.divideAndRound(BigDecimal.java:1439)
at java.math.BigDecimal.divide(BigDecimal.java:1385)
at bigdecimal.OldDivideDemo.main(OldDivideDemo.java:21)
```

“Старые” арифметические операции

```
public class OldSetScaleDemo {

    public static void main(String[] args) {

        BigDecimal a = new BigDecimal("2.333333");

        System.out.println(a);
        System.out.println(a.setScale(4, BigDecimal.ROUND_DOWN));
        System.out.println(a.setScale(3, BigDecimal.ROUND_DOWN));
        System.out.println(a.setScale(2, BigDecimal.ROUND_DOWN));
        System.out.println(a.setScale(1, BigDecimal.ROUND_DOWN));
        System.out.println(a.setScale(0, BigDecimal.ROUND_DOWN));
    }
}
```

```
2.333333
2.3333
2.333
2.33
2.3
2
```

Преобразования к другим типам

Преобразования к другим типам

```
public abstract class Number implements java.io.Serializable {

    public abstract int intValue();
    public abstract long longValue();

    public abstract float floatValue();
    public abstract double doubleValue();

    public byte byteValue() {
        return (byte)intValue();
    }

    public short shortValue() {
        return (short)intValue();
    }
}
```

Преобразования к BigInteger

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {


    public BigInteger toBigInteger() {
        return this.setScale(0, ROUND_DOWN).inflate();
    }

    public BigInteger toBigIntegerExact() {
        return this.setScale(0, ROUND_UNNECESSARY).inflate();
    }
}
```

Преобразования к BigInteger

```
public class ToBigIntegerDemo {

    public static void main(String[] args) {

        BigDecimal decA =  new BigDecimal("123.456789");
        System.out.println("BigDecimal value is: " + decA);

        BigInteger intA = decA.toBigInteger();
        System.out.println("Converted BigInteger value is: " + intA);

        intA = decA.toBigIntegerExact();
        System.out.println("Converted BigInteger using exact conversion value is: " + intA);
    }
}
```

```
BigDecimal value is: 123.456789
Converted BigInteger value is: 123
Exception in thread "main" java.lang.ArithmeticException: Rounding necessary
at java.math.BigDecimal.divideAndRound(Unknown Source)
at java.math.BigDecimal.setScale(Unknown Source)
at java.math.BigDecimal.toBigIntegerExact(Unknown Source)
at bigdecimal.ConversionDemo.main(ConversionDemo.java:16)
```

Преобразования к примитивным типам

```
public class BigDecimal extends Number implements Comparable<BigDecimal> {

    public float floatValue(){
        if (scale == 0 && intCompact != INFLATED)
            return (float)intCompact;
        return Float.parseFloat(this.toString());
    }

    public double doubleValue(){
        if (scale == 0 && intCompact != INFLATED)
            return (double)intCompact;
        return Double.parseDouble(this.toString());
    }

    public long longValue(){
        return (intCompact != INFLATED && scale == 0) ? intCompact : toBigInteger().longValue();
    }

    public int intValue() {
        return  (intCompact != INFLATED && scale == 0) ? (int)intCompact : toBigInteger().intValue();
    }

    public long longValueExact()
    public int intValueExact()
    public short shortValueExact()
    public byte byteValueExact()
}
```

Преобразования к числам с плавающей точкой

```
public class ToDoubleDemo {

    public static void main(String[] args) {

        BigDecimal decA = new BigDecimal("1.23456789012345678 9");
        System.out.println("BigDecimal value is: " + decA);

        double doubleA = decA.doubleValue();
        System.out.println("Converted double value is: " + doubleA);

        float floatA = decA.floatValue();
        System.out.println("Converted float value is: " + floatA);
    }
}
```

```
BigDecimal value is: 1.234567890123456789
Converted double value is: 1.2345678901234567
Converted float value is: 1.2345679
```

Преобразования к целым числам

```
public class ToIntDemo {

    public static void main(String[] args) {

        BigDecimal decA = new BigDecimal("12345678901234567890.0123456789");
        System.out.println("BigDecimal value is: " + decA);

        long longA = decA.longValue();
        System.out.println("Converted long value is: " + longA);

        longA = decA.longValueExact();
        System.out.println("Converted long value using exact conversion is: " + longA);
    }
}
```

```
BigDecimal value is: 12345678901234567890.0123456789
Converted long value is: 12345678901234567890
Exception in thread "main" java.lang.ArithmeticException: Rounding necessary
at java.math.BigDecimal.divideAndRound(Unknown Source)
at java.math.BigDecimal.setScale(Unknown Source)
at java.math.BigDecimal.longValueExact(Unknown Source)
at bigdecimal.ToIntDemo.main(ToIntDemo.java:16)
```