compact1, compact2, compact3

java.math

Class BigInteger

java.lang.Object
    java.lang.Number
        java.math.BigInteger

All Implemented Interfaces:

`Serializable, Comparable<BigInteger>`

```
public class BigInteger
extends Number
implements Comparable<BigInteger>
```

Immutable arbitrary-precision integers. All operations behave as if BigIntegers were represented in two's-complement notation (like Java's primitive integer types). BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from java.lang.Math. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations.

Semantics of arithmetic operations exactly mimic those of Java's integer arithmetic operators, as defined in *The Java Language Specification*. For example, division by zero throws an `ArithmeticException`, and division of a negative by a positive yields a negative (or zero) remainder. All of the details in the Spec concerning overflow are ignored, as BigIntegers are made as large as necessary to accommodate the results of an operation.

Semantics of shift operations extend those of Java's shift operators to allow for negative shift distances. A right-shift with a negative shift distance results in a left shift, and vice-versa. The unsigned right shift operator (>>>) is omitted, as this operation makes little sense in combination with the "infinite word size" abstraction provided by this class.

Semantics of bitwise logical operations exactly mimic those of Java's bitwise integer operators. The binary operators (`and`, `or`, `xor`) implicitly perform sign extension on the shorter of the two operands prior to performing the operation.

Comparison operations perform signed integer comparisons, analogous to those performed by Java's relational and equality operators.

Modular arithmetic operations are provided to compute residues, perform exponentiation, and compute multiplicative inverses. These methods always return a non-negative result, between `0` and `(modulus - 1)`, inclusive.

Bit operations operate on a single bit of the two's-complement representation of their operand. If necessary, the operand is sign- extended so that it contains the designated bit. None of the single-bit operations can produce a BigInteger with a different sign from the BigInteger being operated on, as they affect only a single bit, and the "infinite word size" abstraction provided by this class ensures that there are infinitely many "virtual sign bits" preceding each BigInteger.

For the sake of brevity and clarity, pseudo-code is used throughout the descriptions of BigInteger methods. The pseudo-code expression `(i + j)` is shorthand for "a BigInteger whose value is that of the BigInteger `i` plus that of the BigInteger `j`." The pseudo-code expression `(i == j)` is shorthand for "`true` if and only if the BigInteger `i` represents the same value as the BigInteger `j`." Other pseudo-code expressions are interpreted similarly.

All methods and constructors in this class throw `NullPointerException` when passed a null object reference for any input parameter. BigInteger must support values in the range $-2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) to $+2^{\texttt{Integer.MAX\_VALUE}}$ (exclusive) and may support values outside of that range. The range of probable prime values is limited and may be less than the full supported positive range of `BigInteger`. The range must be at least 1 to $2^{500000000}$.

Implementation Note:

`BigInteger constructors and operations throw ArithmeticException when the result is out of the supported range of` $-2^{\texttt{Integer.MAX\_VALUE}}$ `(exclusive) to` $+2^{\texttt{Integer.MAX\_VALUE}}$ `(exclusive).`

Since:

`JDK1.1`

See Also:

`BigDecimal, Serialized Form`

Field Summary

Fields

Modifier and Type	Field and Description
`static BigInteger`	**`ONE`** The BigInteger constant one.
`static BigInteger`	**`TEN`** The BigInteger constant ten.
`static BigInteger`	**`ZERO`** The BigInteger constant zero.

Constructor Summary

Constructors

Constructor and Description

BigInteger(byte[] val)
Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger.

BigInteger(int signum, byte[] magnitude)
Translates the sign-magnitude representation of a BigInteger into a BigInteger.

BigInteger(int bitLength, int certainty, **Random** rnd)
Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.

BigInteger(int numBits, **Random** rnd)
Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to ($2^{numBits} - 1$), inclusive.

BigInteger(**String** val)
Translates the decimal String representation of a BigInteger into a BigInteger.

BigInteger(**String** val, int radix)
Translates the String representation of a BigInteger in the specified radix into a BigInteger.

Method Summary

All Methods Static Methods Instance Methods Concrete Methods

Modifier and Type	Method and Description
BigInteger	**abs**() Returns a BigInteger whose value is the absolute value of this BigInteger.
BigInteger	**add**(**BigInteger** val) Returns a BigInteger whose value is (this + val).
BigInteger	**and**(**BigInteger** val) Returns a BigInteger whose value is (this & val).
BigInteger	**andNot**(**BigInteger** val) Returns a BigInteger whose value is (this & ~val).
int	**bitCount**() Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
int	**bitLength**() Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit.
byte	**byteValueExact**()

	Converts this `BigInteger` to a `byte`, checking for lost information.
BigInteger	**clearBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
int	**compareTo**(**BigInteger** val) Compares this BigInteger with the specified BigInteger.
BigInteger	**divide**(**BigInteger** val) Returns a BigInteger whose value is `(this / val)`.
BigInteger[]	**divideAndRemainder**(**BigInteger** val) Returns an array of two BigIntegers containing `(this / val)` followed by `(this % val)`.
double	**doubleValue**() Converts this BigInteger to a `double`.
boolean	**equals**(**Object** x) Compares this BigInteger with the specified Object for equality.
BigInteger	**flipBit**(int n) Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.
float	**floatValue**() Converts this BigInteger to a `float`.
BigInteger	**gcd**(**BigInteger** val) Returns a BigInteger whose value is the greatest common divisor of `abs(this)` and `abs(val)`.
int	**getLowestSetBit**() Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit).
int	**hashCode**() Returns the hash code for this BigInteger.
int	**intValue**() Converts this BigInteger to an `int`.
int	**intValueExact**() Converts this `BigInteger` to an `int`, checking for lost information.
boolean	**isProbablePrime**(int certainty) Returns `true` if this BigInteger is probably prime, `false` if it's definitely composite.
long	**longValue**()

	Converts this BigInteger to a `long`.
`long`	**`longValueExact`**`()` Converts this `BigInteger` to a `long`, checking for lost information.
`BigInteger`	**`max`**`(`**`BigInteger`** `val)` Returns the maximum of this BigInteger and `val`.
`BigInteger`	**`min`**`(`**`BigInteger`** `val)` Returns the minimum of this BigInteger and `val`.
`BigInteger`	**`mod`**`(`**`BigInteger`** `m)` Returns a BigInteger whose value is `(this mod m)`.
`BigInteger`	**`modInverse`**`(`**`BigInteger`** `m)` Returns a BigInteger whose value is $(this^{-1} \bmod m)$.
`BigInteger`	**`modPow`**`(`**`BigInteger`** `exponent, `**`BigInteger`** `m)` Returns a BigInteger whose value is $(this^{exponent} \bmod m)$.
`BigInteger`	**`multiply`**`(`**`BigInteger`** `val)` Returns a BigInteger whose value is `(this * val)`.
`BigInteger`	**`negate`**`()` Returns a BigInteger whose value is `(-this)`.
`BigInteger`	**`nextProbablePrime`**`()` Returns the first integer greater than this `BigInteger` that is probably prime.
`BigInteger`	**`not`**`()` Returns a BigInteger whose value is `(~this)`.
`BigInteger`	**`or`**`(`**`BigInteger`** `val)` Returns a BigInteger whose value is `(this | val)`.
`BigInteger`	**`pow`**`(int exponent)` Returns a BigInteger whose value is $(this^{exponent})$.
`static` **`BigInteger`**	**`probablePrime`**`(int bitLength, `**`Random`** `rnd)` Returns a positive BigInteger that is probably prime, with the specified bitLength.
`BigInteger`	**`remainder`**`(`**`BigInteger`** `val)` Returns a BigInteger whose value is `(this % val)`.
`BigInteger`	**`setBit`**`(int n)` Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
`BigInteger`	**`shiftLeft`**`(int n)` Returns a BigInteger whose value is `(this << n)`.

BigInteger	**shiftRight**(int n) Returns a BigInteger whose value is (this >> n).
short	**shortValueExact**() Converts this BigInteger to a short, checking for lost information.
int	**signum**() Returns the signum function of this BigInteger.
BigInteger	**subtract**(**BigInteger** val) Returns a BigInteger whose value is (this - val).
boolean	**testBit**(int n) Returns true if and only if the designated bit is set.
byte[]	**toByteArray**() Returns a byte array containing the two's-complement representation of this BigInteger.
String	**toString**() Returns the decimal String representation of this BigInteger.
String	**toString**(int radix) Returns the String representation of this BigInteger in the given radix.
static **BigInteger**	**valueOf**(long val) Returns a BigInteger whose value is equal to that of the specified long.
BigInteger	**xor**(**BigInteger** val) Returns a BigInteger whose value is (this ^ val).

Methods inherited from class java.lang.Number

byteValue, shortValue

Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Field Detail

ZERO

```
public static final BigInteger ZERO
```

The BigInteger constant zero.

Since:

1.2

ONE

```
public static final BigInteger ONE
```

The BigInteger constant one.

Since:

1.2

TEN

```
public static final BigInteger TEN
```

The BigInteger constant ten.

Since:

1.5

Constructor Detail

BigInteger

```
public BigInteger(byte[] val)
```

Translates a byte array containing the two's-complement binary representation of a BigInteger into a BigInteger. The input array is assumed to be in *big-endian* byte-order: the most significant byte is in the zeroth element.

Parameters:

`val - big-endian two's-complement binary representation of BigInteger.`

Throws:

`NumberFormatException - val is zero bytes long.`

BigInteger

```
public BigInteger(int signum,
                  byte[] magnitude)
```

Translates the sign-magnitude representation of a BigInteger into a BigInteger. The sign is represented as an integer signum value: -1 for negative, 0 for zero, or 1 for positive. The magnitude is a byte array in *big-endian* byte-order: the most significant byte is in the

zeroth element. A zero-length magnitude array is permissible, and will result in a BigInteger value of 0, whether signum is -1, 0 or 1.

Parameters:

`signum - signum of the number (-1 for negative, 0 for zero, 1 for positive).`

`magnitude - big-endian binary representation of the magnitude of the number.`

Throws:

`NumberFormatException - signum is not one of the three legal values (-1, 0, and 1), or signum is 0 and magnitude contains one or more non-zero bytes.`

BigInteger

```
public BigInteger(String val,
                  int radix)
```

Translates the String representation of a BigInteger in the specified radix into a BigInteger. The String representation consists of an optional minus or plus sign followed by a sequence of one or more digits in the specified radix. The character-to-digit mapping is provided by `Character.digit`. The String may not contain any extraneous characters (whitespace, for example).

Parameters:

`val - String representation of BigInteger.`

`radix - radix to be used in interpreting val.`

Throws:

`NumberFormatException - val is not a valid representation of a BigInteger in the specified radix, or radix is outside the range from Character.MIN_RADIX to Character.MAX_RADIX, inclusive.`

See Also:

`Character.digit(char, int)`

BigInteger

```
public BigInteger(String val)
```

Translates the decimal String representation of a BigInteger into a BigInteger. The String representation consists of an optional minus sign followed by a sequence of one or more decimal digits. The character-to-digit mapping is provided by `Character.digit`. The String may not contain any extraneous characters (whitespace, for example).

Parameters:

`val - decimal String representation of BigInteger.`

Throws:

`NumberFormatException - val is not a valid representation of a BigInteger.`

See Also:

Character.digit(char, int)

BigInteger

```
public BigInteger(int numBits,
                  Random rnd)
```

Constructs a randomly generated BigInteger, uniformly distributed over the range 0 to ($2^{numBits} - 1$), inclusive. The uniformity of the distribution assumes that a fair source of random bits is provided in rnd. Note that this constructor always constructs a non-negative BigInteger.

Parameters:

numBits - maximum bitLength of the new BigInteger.

rnd - source of randomness to be used in computing the new BigInteger.

Throws:

IllegalArgumentException - numBits is negative.

See Also:

bitLength()

BigInteger

```
public BigInteger(int bitLength,
                  int certainty,
                  Random rnd)
```

Constructs a randomly generated positive BigInteger that is probably prime, with the specified bitLength.

It is recommended that the probablePrime method be used in preference to this constructor unless there is a compelling need to specify a certainty.

Parameters:

bitLength - bitLength of the returned BigInteger.

certainty - a measure of the uncertainty that the caller is willing to tolerate. The probability that the new BigInteger represents a prime number will exceed ($1 - 1/2^{certainty}$). The execution time of this constructor is proportional to the value of this parameter.

rnd - source of random bits used to select candidates to be tested for primality.

Throws:

ArithmeticException - bitLength < 2 or bitLength is too large.

See Also:

bitLength()

Method Detail

probablePrime

```
public static BigInteger probablePrime(int bitLength,
                                       Random rnd)
```

Returns a positive BigInteger that is probably prime, with the specified bitLength. The probability that a BigInteger returned by this method is composite does not exceed 2^{-100}.

Parameters:

bitLength - bitLength of the returned BigInteger.

rnd - source of random bits used to select candidates to be tested for primality.

Returns:

a BigInteger of bitLength bits that is probably prime

Throws:

ArithmeticException - bitLength < 2 or bitLength is too large.

Since:

1.4

See Also:

bitLength()

nextProbablePrime

```
public BigInteger nextProbablePrime()
```

Returns the first integer greater than this BigInteger that is probably prime. The probability that the number returned by this method is composite does not exceed 2^{-100}. This method will never skip over a prime when searching: if it returns p, there is no prime q such that this < q < p.

Returns:

the first integer greater than this BigInteger that is probably prime.

Throws:

ArithmeticException - this < 0 or this is too large.

Since:

1.5

valueOf

```
public static BigInteger valueOf(long val)
```

Returns a BigInteger whose value is equal to that of the specified `long`. This "static factory method" is provided in preference to a (`long`) constructor because it allows for reuse of frequently used BigIntegers.

Parameters:

`val - value of the BigInteger to return.`

Returns:

`a BigInteger with the specified value.`

add

```
public BigInteger add(BigInteger val)
```

Returns a BigInteger whose value is `(this + val)`.

Parameters:

`val - value to be added to this BigInteger.`

Returns:

`this + val`

subtract

```
public BigInteger subtract(BigInteger val)
```

Returns a BigInteger whose value is `(this - val)`.

Parameters:

`val - value to be subtracted from this BigInteger.`

Returns:

`this - val`

multiply

```
public BigInteger multiply(BigInteger val)
```

Returns a BigInteger whose value is `(this * val)`.

Implementation Note:

`An implementation may offer better algorithmic performance when val == this.`

Parameters:

`val - value to be multiplied by this BigInteger.`

Returns:

`this * val`

divide

```
public BigInteger divide(BigInteger val)
```

Returns a BigInteger whose value is `(this / val)`.

Parameters:

`val - value by which this BigInteger is to be divided.`

Returns:

`this / val`

Throws:

`ArithmeticException - if val is zero.`

divideAndRemainder

```
public BigInteger[] divideAndRemainder(BigInteger val)
```

Returns an array of two BigIntegers containing `(this / val)` followed by `(this % val)`.

Parameters:

`val - value by which this BigInteger is to be divided, and the remainder computed.`

Returns:

`an array of two BigIntegers: the quotient (this / val) is the initial element, and the remainder (this % val) is the final element.`

Throws:

`ArithmeticException - if val is zero.`

remainder

```
public BigInteger remainder(BigInteger val)
```

Returns a BigInteger whose value is `(this % val)`.

Parameters:

`val - value by which this BigInteger is to be divided, and the remainder computed.`

Returns:

`this % val`

Throws:

ArithmeticException - if val is zero.

pow

```
public BigInteger pow(int exponent)
```

Returns a BigInteger whose value is ($this^{exponent}$). Note that exponent is an integer rather than a BigInteger.

Parameters:

exponent - exponent to which this BigInteger is to be raised.

Returns:

$this^{exponent}$

Throws:

ArithmeticException - exponent is negative. (This would cause the operation to yield a non-integer value.)

gcd

```
public BigInteger gcd(BigInteger val)
```

Returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val). Returns 0 if this == 0 && val == 0.

Parameters:

val - value with which the GCD is to be computed.

Returns:

GCD(abs(this), abs(val))

abs

```
public BigInteger abs()
```

Returns a BigInteger whose value is the absolute value of this BigInteger.

Returns:

abs(this)

negate

```
public BigInteger negate()
```

Returns a BigInteger whose value is (-this).

Returns:

-this

signum

```
public int signum()
```

Returns the signum function of this BigInteger.

Returns:

-1, 0 or 1 as the value of this BigInteger is negative, zero or positive.

mod

```
public BigInteger mod(BigInteger m)
```

Returns a BigInteger whose value is (this mod m). This method differs from remainder in that it always returns a *non-negative* BigInteger.

Parameters:

m - the modulus.

Returns:

this mod m

Throws:

ArithmeticException - m ≤ 0

See Also:

remainder(java.math.BigInteger)

modPow

```
public BigInteger modPow(BigInteger exponent,
                         BigInteger m)
```

Returns a BigInteger whose value is ($this^{exponent}$ mod m). (Unlike pow, this method permits negative exponents.)

Parameters:

exponent - the exponent.

m - the modulus.

Returns:

$this^{exponent}$ mod m

Throws:

ArithmeticException - m ≤ 0 or the exponent is negative and this BigInteger is not *relatively prime* to m.

See Also:

`modInverse(java.math.BigInteger)`

modInverse

`public BigInteger modInverse(BigInteger m)`

Returns a BigInteger whose value is $(this^{-1} \bmod m)$.

Parameters:

`m - the modulus.`

Returns:

$this^{-1} \bmod m$.

Throws:

`ArithmeticException - m ≤ 0, or this BigInteger has no multiplicative inverse mod m (that is, this BigInteger is not relatively prime to m).`

shiftLeft

`public BigInteger shiftLeft(int n)`

Returns a BigInteger whose value is $(this \ll n)$. The shift distance, n, may be negative, in which case this method performs a right shift. (Computes $floor(this * 2^n)$.)

Parameters:

`n - shift distance, in bits.`

Returns:

$this \ll n$

See Also:

`shiftRight(int)`

shiftRight

`public BigInteger shiftRight(int n)`

Returns a BigInteger whose value is $(this \gg n)$. Sign extension is performed. The shift distance, n, may be negative, in which case this method performs a left shift. (Computes $floor(this / 2^n)$.)

Parameters:

`n - shift distance, in bits.`

Returns:

$this \gg n$

See Also:

shiftLeft(int)

and

```
public BigInteger and(BigInteger val)
```

Returns a BigInteger whose value is (this & val). (This method returns a negative BigInteger if and only if this and val are both negative.)

Parameters:

val - value to be AND'ed with this BigInteger.

Returns:

this & val

or

```
public BigInteger or(BigInteger val)
```

Returns a BigInteger whose value is (this | val). (This method returns a negative BigInteger if and only if either this or val is negative.)

Parameters:

val - value to be OR'ed with this BigInteger.

Returns:

this | val

xor

```
public BigInteger xor(BigInteger val)
```

Returns a BigInteger whose value is (this ^ val). (This method returns a negative BigInteger if and only if exactly one of this and val are negative.)

Parameters:

val - value to be XOR'ed with this BigInteger.

Returns:

this ^ val

not

```
public BigInteger not()
```

Returns a BigInteger whose value is (~this). (This method returns a negative value if and only if this BigInteger is non-negative.)

Returns:

`~this`

andNot

```
public BigInteger andNot(BigInteger val)
```

Returns a BigInteger whose value is `(this & ~val)`. This method, which is equivalent to `and(val.not())`, is provided as a convenience for masking operations. (This method returns a negative BigInteger if and only if `this` is negative and `val` is positive.)

Parameters:

`val - value to be complemented and AND'ed with this BigInteger.`

Returns:

`this & ~val`

testBit

```
public boolean testBit(int n)
```

Returns `true` if and only if the designated bit is set. (Computes `((this & (1<<n)) != 0)`.)

Parameters:

`n - index of bit to test.`

Returns:

`true if and only if the designated bit is set.`

Throws:

`ArithmeticException - n is negative.`

setBit

```
public BigInteger setBit(int n)
```

Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set. (Computes `(this | (1<<n))`.)

Parameters:

`n - index of bit to set.`

Returns:

`this | (1<<n)`

Throws:

`ArithmeticException - n is negative.`

clearBit

```
public BigInteger clearBit(int n)
```

Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared. (Computes `(this & ~(1<<n))`.)

Parameters:

`n - index of bit to clear.`

Returns:

`this & ~(1<<n)`

Throws:

`ArithmeticException - n is negative.`

flipBit

```
public BigInteger flipBit(int n)
```

Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped. (Computes `(this ^ (1<<n))`.)

Parameters:

`n - index of bit to flip.`

Returns:

`this ^ (1<<n)`

Throws:

`ArithmeticException - n is negative.`

getLowestSetBit

```
public int getLowestSetBit()
```

Returns the index of the rightmost (lowest-order) one bit in this BigInteger (the number of zero bits to the right of the rightmost one bit). Returns -1 if this BigInteger contains no one bits. (Computes `(this == 0? -1 : log2(this & -this))`.)

Returns:

`index of the rightmost one bit in this BigInteger.`

bitLength

```
public int bitLength()
```

Returns the number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit. For positive BigIntegers, this is equivalent to the number

of bits in the ordinary binary representation. (Computes `(ceil(log2(this < 0 ? -this : this+1)))`.)

Returns:

number of bits in the minimal two's-complement representation of this BigInteger, *excluding* a sign bit.

bitCount

```
public int bitCount()
```

Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit. This method is useful when implementing bit-vector style sets atop BigIntegers.

Returns:

number of bits in the two's complement representation of this BigInteger that differ from its sign bit.

isProbablePrime

```
public boolean isProbablePrime(int certainty)
```

Returns `true` if this BigInteger is probably prime, `false` if it's definitely composite. If `certainty` is ≤ 0, `true` is returned.

Parameters:

`certainty` - a measure of the uncertainty that the caller is willing to tolerate: if the call returns `true` the probability that this BigInteger is prime exceeds $(1 - 1/2^{certainty})$. The execution time of this method is proportional to the value of this parameter.

Returns:

true if this BigInteger is probably prime, false if it's definitely composite.

compareTo

```
public int compareTo(BigInteger val)
```

Compares this BigInteger with the specified BigInteger. This method is provided in preference to individual methods for each of the six boolean comparison operators (<, ==, >, >=, !=, <=). The suggested idiom for performing these comparisons is: `(x.compareTo(y)` <*op*> `0)`, where <*op*> is one of the six comparison operators.

Specified by:

`compareTo` in interface `Comparable<BigInteger>`

Parameters:

val - BigInteger to which this BigInteger is to be compared.

Returns:

-1, 0 or 1 as this BigInteger is numerically less than, equal to, or greater than val.

equals

```
public boolean equals(Object x)
```

Compares this BigInteger with the specified Object for equality.

Overrides:

equals in class Object

Parameters:

x - Object to which this BigInteger is to be compared.

Returns:

true if and only if the specified Object is a BigInteger whose value is numerically equal to this BigInteger.

See Also:

Object.hashCode(), HashMap

min

```
public BigInteger min(BigInteger val)
```

Returns the minimum of this BigInteger and val.

Parameters:

val - value with which the minimum is to be computed.

Returns:

the BigInteger whose value is the lesser of this BigInteger and val. If they are equal, either may be returned.

max

```
public BigInteger max(BigInteger val)
```

Returns the maximum of this BigInteger and val.

Parameters:

val - value with which the maximum is to be computed.

Returns:

the BigInteger whose value is the greater of this and val. If they are equal, either may be returned.

hashCode

```
public int hashCode()
```

Returns the hash code for this BigInteger.

Overrides:

`hashCode` in class `Object`

Returns:

hash code for this BigInteger.

See Also:

`Object.equals(java.lang.Object)`, `System.identityHashCode(java.lang.Object)`

toString

```
public String toString(int radix)
```

Returns the String representation of this BigInteger in the given radix. If the radix is outside the range from `Character.MIN_RADIX` to `Character.MAX_RADIX` inclusive, it will default to 10 (as is the case for `Integer.toString`). The digit-to-character mapping provided by `Character.forDigit` is used, and a minus sign is prepended if appropriate. (This representation is compatible with the `(String, int)` constructor.)

Parameters:

radix - radix of the String representation.

Returns:

String representation of this BigInteger in the given radix.

See Also:

`Integer.toString(int, int)`, `Character.forDigit(int, int)`, `BigInteger(java.lang.String, int)`

toString

```
public String toString()
```

Returns the decimal String representation of this BigInteger. The digit-to-character mapping provided by `Character.forDigit` is used, and a minus sign is prepended if appropriate. (This representation is compatible with the `(String)` constructor, and allows for String concatenation with Java's + operator.)

Overrides:

`toString` in class `Object`

Returns:

decimal String representation of this BigInteger.

See Also:

Character.forDigit(int, int), BigInteger(java.lang.String)

toByteArray

```
public byte[] toByteArray()
```

Returns a byte array containing the two's-complement representation of this BigInteger. The byte array will be in *big-endian* byte-order: the most significant byte is in the zeroth element. The array will contain the minimum number of bytes required to represent this BigInteger, including at least one sign bit, which is `(ceil((this.bitLength() + 1)/8))`. (This representation is compatible with the `(byte[])` constructor.)

Returns:

a byte array containing the two's-complement representation of this BigInteger.

See Also:

BigInteger(byte[])

intValue

```
public int intValue()
```

Converts this BigInteger to an `int`. This conversion is analogous to a *narrowing primitive conversion* from `long` to `int` as defined in section 5.1.3 of *The Java™ Language Specification*: if this BigInteger is too big to fit in an `int`, only the low-order 32 bits are returned. Note that this conversion can lose information about the overall magnitude of the BigInteger value as well as return a result with the opposite sign.

Specified by:

intValue in class Number

Returns:

this BigInteger converted to an int.

See Also:

intValueExact()

longValue

```
public long longValue()
```

Converts this BigInteger to a `long`. This conversion is analogous to a *narrowing primitive conversion* from `long` to `int` as defined in section 5.1.3 of *The Java™ Language Specification*: if this BigInteger is too big to fit in a `long`, only the low-order 64 bits are

returned. Note that this conversion can lose information about the overall magnitude of the BigInteger value as well as return a result with the opposite sign.

Specified by:

`longValue` in class `Number`

Returns:

this BigInteger converted to a long.

See Also:

`longValueExact()`

floatValue

```
public float floatValue()
```

Converts this BigInteger to a `float`. This conversion is similar to the *narrowing primitive conversion* from `double` to `float` as defined in section 5.1.3 of *The Java™ Language Specification*: if this BigInteger has too great a magnitude to represent as a `float`, it will be converted to `Float.NEGATIVE_INFINITY` or `Float.POSITIVE_INFINITY` as appropriate. Note that even when the return value is finite, this conversion can lose information about the precision of the BigInteger value.

Specified by:

`floatValue` in class `Number`

Returns:

this BigInteger converted to a float.

doubleValue

```
public double doubleValue()
```

Converts this BigInteger to a `double`. This conversion is similar to the *narrowing primitive conversion* from `double` to `float` as defined in section 5.1.3 of *The Java™ Language Specification*: if this BigInteger has too great a magnitude to represent as a `double`, it will be converted to `Double.NEGATIVE_INFINITY` or `Double.POSITIVE_INFINITY` as appropriate. Note that even when the return value is finite, this conversion can lose information about the precision of the BigInteger value.

Specified by:

`doubleValue` in class `Number`

Returns:

this BigInteger converted to a double.

longValueExact

```
public long longValueExact()
```

Converts this BigInteger to a long, checking for lost information. If the value of this BigInteger is out of the range of the long type, then an ArithmeticException is thrown.

Returns:

this BigInteger converted to a long.

Throws:

ArithmeticException - if the value of this will not exactly fit in a long.

Since:

1.8

See Also:

longValue()

intValueExact

```
public int intValueExact()
```

Converts this BigInteger to an int, checking for lost information. If the value of this BigInteger is out of the range of the int type, then an ArithmeticException is thrown.

Returns:

this BigInteger converted to an int.

Throws:

ArithmeticException - if the value of this will not exactly fit in a int.

Since:

1.8

See Also:

intValue()

shortValueExact

```
public short shortValueExact()
```

Converts this BigInteger to a short, checking for lost information. If the value of this BigInteger is out of the range of the short type, then an ArithmeticException is thrown.

Returns:

this BigInteger converted to a short.

Throws:

ArithmeticException - if the value of this will not exactly fit in a short.

Since:

1.8

See Also:

Number.shortValue()

byteValueExact

```
public byte byteValueExact()
```

Converts this `BigInteger` to a `byte`, checking for lost information. If the value of this `BigInteger` is out of the range of the `byte` type, then an `ArithmeticException` is thrown.

Returns:

this BigInteger converted to a byte.

Throws:

`ArithmeticException` - if the value of this will not exactly fit in a byte.

Since:

1.8

See Also:

Number.byteValue()