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 (>>>)

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.

Since:

JDK1.1

See Also:

BigDecimal, Serialized Form

Field Summary

<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifier and Type</td>
</tr>
<tr>
<td>static BigInteger</td>
</tr>
<tr>
<td>static BigInteger</td>
</tr>
<tr>
<td>static BigInteger</td>
</tr>
</tbody>
</table>

Constructor Summary

<table>
<thead>
<tr>
<th>Constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor and Description</td>
</tr>
<tr>
<td>BigInteger(byte[] val)</td>
</tr>
<tr>
<td>BigInteger(int signum, byte[] magnitude)</td>
</tr>
<tr>
<td>BigInteger(int bitLength, int certainty, Random rnd)</td>
</tr>
<tr>
<td>BigInteger(int numBits, Random rnd)</td>
</tr>
<tr>
<td>BigInteger(String val)</td>
</tr>
<tr>
<td>BigInteger(String val, int radix)</td>
</tr>
</tbody>
</table>

Method Summary
<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BigInteger</td>
<td>abs()</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is the absolute value of this BigInteger.</td>
</tr>
<tr>
<td>BigInteger</td>
<td>add(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is (this + val).</td>
</tr>
<tr>
<td>BigInteger</td>
<td>and(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is (this &amp; val).</td>
</tr>
<tr>
<td>BigInteger</td>
<td>andNot(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is (this &amp; ~val).</td>
</tr>
<tr>
<td>int</td>
<td>bitCount()</td>
</tr>
<tr>
<td></td>
<td>Returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.</td>
</tr>
<tr>
<td>int</td>
<td>bitLength()</td>
</tr>
<tr>
<td></td>
<td>Returns the number of bits in the minimal two's-complement representation of this BigInteger, excluding a sign bit.</td>
</tr>
<tr>
<td>BigInteger</td>
<td>clearBit(int n)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.</td>
</tr>
<tr>
<td>int</td>
<td>compareTo(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Compares this BigInteger with the specified BigInteger.</td>
</tr>
<tr>
<td>BigInteger</td>
<td>divide(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is (this / val).</td>
</tr>
<tr>
<td>BigInteger[]</td>
<td>divideAndRemainder(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Returns an array of two BigIntegers containing (this / val) followed by (this % val).</td>
</tr>
<tr>
<td>double</td>
<td>doubleValue()</td>
</tr>
<tr>
<td></td>
<td>Converts this BigInteger to a double.</td>
</tr>
<tr>
<td>boolean</td>
<td>equals(Object x)</td>
</tr>
<tr>
<td></td>
<td>Compares this BigInteger with the specified Object for equality.</td>
</tr>
<tr>
<td>BigInteger</td>
<td>flipBit(int n)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.</td>
</tr>
<tr>
<td>float</td>
<td>floatValue()</td>
</tr>
<tr>
<td></td>
<td>Converts this BigInteger to a float.</td>
</tr>
<tr>
<td>BigInteger</td>
<td>gcd(BigInteger val)</td>
</tr>
<tr>
<td></td>
<td>Returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val).</td>
</tr>
<tr>
<td>int</td>
<td>getLowestSetBit()</td>
</tr>
<tr>
<td></td>
<td>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).</td>
</tr>
<tr>
<td>int</td>
<td>hashCode()</td>
</tr>
<tr>
<td></td>
<td>Returns the hash code for this BigInteger.</td>
</tr>
<tr>
<td>int</td>
<td>intValue()</td>
</tr>
<tr>
<td></td>
<td>Converts this BigInteger to an int.</td>
</tr>
</tbody>
</table>
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.

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\(^{-1}\) mod m).

BigInteger modPow(BigInteger exponent, BigInteger m)
Returns a BigInteger whose value is (this\(^{exponent}\) mod 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).

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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] toByteArray()</td>
<td>Returns a byte array containing the two's-complement representation of this</td>
</tr>
<tr>
<td></td>
<td>BigInteger.</td>
</tr>
<tr>
<td>String toString()</td>
<td>Returns the decimal String representation of this BigInteger.</td>
</tr>
<tr>
<td>String toString(int radix)</td>
<td>Returns the String representation of this BigInteger in the given radix.</td>
</tr>
<tr>
<td>static BigInteger valueOf(long val)</td>
<td>Returns a BigInteger whose value is equal to that of the specified long.</td>
</tr>
<tr>
<td>BigInteger xor(BigInteger val)</td>
<td>Returns a BigInteger whose value is ((\text{this} \oplus \text{val})).</td>
</tr>
</tbody>
</table>

### 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

```java
public static final BigInteger ZERO
```

The BigInteger constant zero.

**Since:**

1.2

#### ONE

```java
public static final BigInteger ONE
```

The BigInteger constant one.

**Since:**

1.2

#### TEN

```java
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 \((2^{\text{numBits}} - 1)\), 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 \((1 - 1/2^{\text{certainty}})\). 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.

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