#### BigInteger

BigInteger represents immutable arbitrary-precision integers. It is similar to the primitive integer types but allows arbitrary large values.

It is used when integers involved are larger than the limit of long type. For example, the factorial of 50 is 30414093201713378043612608166064768844377641568960512000000000000. This value is too big for an int or long data type to handle. It can only be stored in a BigInteger variable.

It is widely used in security and cryptography applications.

import java.math.BigInteger;

|  |
| --- |
| package flowControl;   *//must import* import java.math.BigInteger;  public class WrapperClass {    public static void main(String[] args) {   BigInteger biFromString = new BigInteger("1234567890987654321");  BigInteger biFromByteArray = new BigInteger(new byte[] { 64, 64, 64, 64, 64, 64 });  BigInteger biFromSignMagnitude = new BigInteger(-1, new byte[] { 64, 64, 64, 64, 64, 64 });   BigInteger i = new BigInteger("123456789012345678901234567890");  BigInteger j = new BigInteger("123456789012345678901234567891");  BigInteger k = new BigInteger("123456789012345678901234567892");   BigInteger sum = i.add(j);  BigInteger difference = i.subtract(j);  BigInteger quotient = i.divide(j);  BigInteger product = i.multiply(j);   BigInteger and = i.and(j);  BigInteger or = i.or(j);  BigInteger not = j.not();  BigInteger xor = i.xor(j);  BigInteger andNot = i.andNot(j);  BigInteger shiftLeft = i.shiftLeft(1);  BigInteger shiftRight = i.shiftRight(1);  BigInteger gcd = j.gcd(k);  BigInteger multiplyAndmod = j.multiply(k).mod(i);  BigInteger modInverse = j.modInverse(i);  BigInteger modPow = j.modPow(k, i);   int bitCount = i.bitCount();  int bitLength = i.bitLength();  int getLowestSetBit = i.getLowestSetBit();  boolean testBit3 = i.testBit(3);  BigInteger setBit12 = i.setBit(12);  BigInteger flipBit0 = i.flipBit(0);  BigInteger clearBit3 = i.clearBit(3);    } } |

#### BigDecimal

BigDecimal represents an immutable arbitrary-precision signed decimal number. It consists of two parts:

Unscaled value – an arbitrary precision integer

Scale – a 32-bit integer representing the number of digits to the right of the decimal point

For example, the BigDecimal 3.14 has the unscaled value of 314 and the scale of 2.

We use BigDecimal for high-precision arithmetic. We also use it for calculations requiring control over scale and rounding off behavior. One such example is calculations involving financial transactions.

We can create a BigDecimal object from String, character array, int, long, and BigInteger:

import java.math.BigDecimal;

|  |
| --- |
| package flowControl;   *//must import* import java.math.BigDecimal; import java.math.BigInteger; import java.util.Random;  public class WrapperClass {    public static void main(String[] args) {   BigDecimal bdFromString = new BigDecimal("0.1");  BigDecimal bdFromCharArray = new BigDecimal(new char[] {'3','.','1','6','1','5'});  BigDecimal bdlFromInt = new BigDecimal(42);  BigDecimal bdFromLong = new BigDecimal(123412345678901L);  BigInteger bigInteger = BigInteger.*probablePrime*(100, new Random());  BigDecimal bdFromBigInteger = new BigDecimal(bigInteger);  BigDecimal bdFromDouble = new BigDecimal(0.1d);   BigDecimal bdFromLong1 = BigDecimal.*valueOf*(123412345678901L);  BigDecimal bdFromLong2 = BigDecimal.*valueOf*(123412345678901L, 2);  BigDecimal bdFromDouble2 = BigDecimal.*valueOf*(0.1d);   BigDecimal bd1 = new BigDecimal("4.0");  BigDecimal bd2 = new BigDecimal("2.0");   BigDecimal sum = bd1.add(bd2);  BigDecimal difference = bd1.subtract(bd2);  BigDecimal quotient = bd1.divide(bd2);  BigDecimal product = bd1.multiply(bd2);     } } |