In Java, there are primitive data types such as int and long to handle integer-related operations. However, there is a limit to the number that these data types can store. For example, the int data type in java can store values from **-2147483648 to 2147483647**, while the long data type can store values from **-9223372036854775808 to 9223372036854775808**.

BigInteger is a class in Java that can store values bigger than what primitive data types can store.

**Introduction to Java BigInteger**

Let's say you are building a factorial calculator that gives a factorial of any number that the user inputs. If the user inputs 5, then it gives output 120 as 5! = 120. Now let's say the user inputs 13, the answer should be 6227020800, however, your program will break if you have used int data type to store the answer. This is because the int data type can only store values up to 2147483647, and as you can see 13! is greater than what the int data type can store. However, if you used a long variable to store the answer, then your program will work. However, if a user inputs a bigger number such as 20 or 100, then again your program will break as the answer will become greater than what primitive data types can store.

To counter the above problem, you can use a Biginteger class in java. Biginteger class in java is used for mathematical operations that require integer values larger than what primitive data types can store. A biginteger class represents immutable arbitrary-precision integers. Apart from storing very large numbers, one can even perform operations on large numbers using biginteger in java.

**Working of BigInteger Class in Java**

In Java, all the bytes are stored in 2's complement method. The first bit determines whether the number is positive or negative.

**BigInteger** class in java used the mag property to store numbers that excess storage capacity of primitive data types. What does this property mean? Well, it means that the Biginteger class in java stores values in an array using the binary representation. The BigInteger java groups the binary representation in 32-bit portions. This removes the limitation of the number of storage possessed by other data types in java.

**BigInteger** works with both numbers and strings. You can store a very large number converted to a string using a biginteger in java.

**Constructors**

**BigInteger** class has constructors which you can use to store numbers or strings in it. Let's have a look at some of these constructors and understand what they are used for.

| **Constructor** | **Meaning** |
| --- | --- |
| BigInteger(byte[] val) | It converts a given byte array containing two's complements of a BigInteger into a BigInteger. |
| BigInteger(int signum, byte[] magnitude) | Signum is used to define the sign of BigIntger, whether it is positive or negative. This constructor converts a sign byte array into a BigInteger. |
| BigInteger(int bitLength, int certainty, Random rnd) | It generates a BigInteger number of the given length. The number generated is usually a prime number. |
| BigInteger(int numBits, Random rnd) | It generates a random integer BigIntger however it is uniformly distributed in the range of (2^numBits-1). |
| BigInteger(String val) | It converts the given string value into a BigIntger value. The given string value should contain only numbers and signs. |
| BigInteger(String val, int radix) | It converts the given string value in the specified radix into a BigInteger value. |

These constructor methods are widely used when declaring BigInteger object/variables. While using BigInteger, you would be following one of the above-given constructor methods to create a BigInteger variable and work with it. Now let's see some in-built methods that BigInteger provides in java that we can use.

**Methods of Java BigInteger Class**

Once you have declared BigInteger class and stored value in it, you will perform certain operations on it. You can add a certain number to BigInteger class declared, multiply a number or even use **Bitwise** operators on BigInteger class. These

For most of these operations, there are methods available in java that you can simply call. Let's have a look at some methods of the Java BigInteger class.

| **Method** | **What it does** |
| --- | --- |
| add(BigInteger val) | Add the given BigInteger value to your current BigInteger Vaule and return the sum. |
| abs() | Return the absolute value of a BigInteger. |
| andNot(BigInteger val) | Perform NAND Operation between the given BigInteger value and your current BigInteger value and returns the output. |
| bitCount() | Return the number of bits that differ from it's sign bit in two's complement. |
| bitLength() | Returns the number of bits in two's complement excluding the sign bit. |
| byteValueExact() | Used for checking lost Information. Converts your BigInteger into byte value. |
| clearBit(int n) | Returns a BigInteger whose value is equal to modified bit. |
| compareTo(BigInteger val) | Compares two BigInteger, the current one vs the one your provide. |
| divide(BigInteger val) | Returns a BigInteger whose value is your current BigInteger divided the given BigInteger value. |
| divideAndRemainder(BigInteger val) | Returns an array of BigInteger of size two. The first position contains the BigInteger whose value is your current BigInteger divided the given BigInteger value. The second positiong contains the remainder of you get when you divide your current BigInteger by given BigInteger value. |
| doubleValue() | Convert your current BigInteger value into a double value. |
| equals(Object x) | Checks and compare whether your BigInteger is equal to given object. |
| flipBit(int n) | Returns a given BigInteger value you get when you flip the given bit of your current BigInteger value. |
| floatValue() | Convert your current BigInteger value into a float value. |
| gcd(BigInteger val) | Returns a BigInteger value which is gcd of your BigInteger value and given BigInteger value. |
| getLowestSetBit() | Return the index of right-most one bit. |
| hashCode() | Returns hash code of your BigInteger value. |
| intValue() | Converts your BigInteger value into int value. |
| intValueExact() | Usually used to check for lost information. Converts the given BigInteger value into int. |
| isProbablePrime(int certainty) | Returns true if your BigInteger value is prime. If not, it returns false. |
| longValue() | Converts your BigInteger value into long value. |
| longValueExact() | Usually used to check for lost information. Converts your BigInteger value into long value. |
| max(BigInteger val) | Returns the maximum BigInteger between your BigInteger value and given BigInteger value. |
| min(BigInteger val) | Returns the minimum BigInteger between your BigInteger value and given BigInteger value. |
| mod(BigInteger m | Returns remainder achieved when you divide your BigInteger value with given BigInteger value. |
| modInverse(BigInteger m) | Returns remainder achieved when your divide your BigInteger value-1 with given BigInteger value. |
| modPow(BigInteger exponent, BigInteger m | Returns remainder achieved when you divide your exponent BigInteger value with given BigInteger value. |
| multiply(BigInteger val) | Returns Product of your BigInteger value and given BigInteger value. |
| negate() | Returns Inverse of your BigInteger value. |
| nextProbablePrime() | Returns the closest Prime Number to your BigInteger value (excluding your current BigInteger value if it's prime.) |
| not() | Returns a BigInteger whose value is NOT of your BigInteger value(~BigIntger value). |
| or(BigInteger val) | Performs OR Operation between your BigInteger value and given BigInteger value. Returns the answer. |
| pow(int exponent) | Returns BigInteger whose value is pow(BigInteger,exponent). |
| probablePrime(int bitLength, Random rnd) | Returns a randm BigInteger whose value is prime and is of given bit length. |
| remainder(BigInteger val) | Returns remainder you get by dividing your BigINteger value by given BigINteger value. |
| setBit(int n) | Returns a BigInteger whose value is your current BigInteger value, with the current bit set. |
| shiftLeft(int n) | Returns a BigInteger after shifting n digits to left. |
| shiftRight(int n) | Returns a BigInteger after shifting n digits to right. |
| shortValueExact() | Usually used to check for lost information. Converts your BigInteger value into a short value. |
| signum() | Returns the signum function of your BigInteger value. |
| sqrt() | Returns square root of your BigInteger value. |
| sqrtAndRemainder() | Returns an array of BigInteger of size two. The first position contains the BigInteger whose value is your square root of your current BigInteger value. The second positiong contains the remainder of you get when you divide your current BigInteger by given square of square root of your BigInteger value. |
| subtract(BigInteger val) | Returns difference between your BigInteger value and given BigInteger value. |
| testBit(int n) | Returns true if the given bit is set in your BigInteger. |
| toByteArray() | Returns a byte array of two's complement of your BigInteger value. It shows how BigInteger stores large values. |
| toString() | Convert your BigInteger value to string and returns it. |
| toString(int radix) | Convert your BigInteger value to string in given radix. |
| valueOf(long val) | Returns a BigInteger whose value is equal to the given long value. |
| xor(BigInteger val) | Performs XOR operation between your BigInteger value and given BigInteger value. The answer is returned to user. |

### Example 1: Create a function in Java That Finds the Factorial of a Number.

Now the question might look simple, but it isn't. It might seem simple since all you need to do is use a for loop. But we are not given any limit on the number that can be given as input. It can be a large number such as **100 or 1000**, in which the limit will **overflow**. So we will use BigInteger to write this function.

### Code

import java.math.BigInteger; *// Importing necessary libraries*

import java.util.Scanner;

public class Example {

*// Method to calculate factorial of a number*

public static BigInteger factorial(int n) {

BigInteger factorialResult = BigInteger.ONE; *// Initializing the result with 1*

*// Looping through all the numbers from 1 to n*

for (int i=1; i<=n; i++) {

factorialResult = factorialResult.multiply(BigInteger.valueOf(i)); *// Multiplying the result with current number*

}

return factorialResult; *// Returning the final factorial result*

}

public static void main(String args[]) throws Exception{

int n = 10; *// Taking a sample input*

System.out.println(factorial(n)); *// Printing the factorial of given input*

}

}

**Output**

**3628800**

The above factorial function will work for any number greater than or equal to 0. It can be 1 or 100 or even 100000.

### Example 2: Create a function that takes two strings and finds returns a string that is the Product of two strings.

The above question is tricky in many ways. **First**, you are given strings and not numbers. **Second**, you are not specified about the length of the string. The string can be 100 digits big in length.

To tackle this question we will use BigInteger. BigInteger allows users to store values both integer as well as string. So we will convert the string into BigInteger, then find the product using BigInteger methods and finally convert the BigInteger back to String and return it.

### Code

import java.math.BigInteger; *// Importing necessary libraries*

import java.util.Scanner;

public class Example {

*// Method to multiply two strings representing numbers*

public static String multiplyString(String num1, String num2) {

BigInteger number1 = new BigInteger(num1); *// Convert the first number string to BigInteger*

BigInteger number2 = new BigInteger(num2); *// Convert the second number string to BigInteger*

BigInteger product = number1.multiply(number2); *// Multiply the two BigIntegers to get the product*

return product.toString(); *// Convert the BigInteger product to string and return it*

}

public static void main(String args[]) throws Exception{

System.out.println(multiplyString("99","100"));

*// Call the multiplyString method with sample inputs and print the result*

}

}

**Output**

**9900**

**Initialization is as follows:**

a = 54;

b = 23;

**A = BigInteger.valueOf(54);**

**B = BigInteger.valueOf(37);**

And for Integers available as strings you can initialize them as follows:

A = new BigInteger(“54”);

B = new BigInteger(“123456789123456789”);

Some constants are also defined in BigInteger class for ease of initialization as follows:

A = BigInteger.ONE;

// Other than this, available constant are BigInteger.ZERO

// and BigInteger.TEN

Mathematical operations are as follows:

int c = a + b;

BigInteger C = A.add(B);

Other similar functions are subtract(), multiply(), divide(), remainder(), but all these functions take BigInteger as their argument so if we want this operation with integers or string convert them to BigInteger before passing them to functions as shown below:

String str = “123456789”;

BigInteger C = A.add(new BigInteger(str));

int val = 123456789;

BigInteger C = A.add(BigInteger.valueOf(val));

Extraction of value from BigInteger is as follows:

int x = A.intValue(); // value should be in limit of int x

long y = A.longValue(); // value should be in limit of long y

String z = A.toString();

Comparison

if (a < b) {} // For primitive int

if (A.compareTo(B) < 0) {} // For BigInteger

Actually compareTo returns -1(less than), 0(Equal), 1(greater than) according to values. For equality we can also use:

if (A.equals(B)) {} // A is equal to B