BigInteger Class in Java

BigInteger class is used for mathematical operation which involves very big integer calculations that are outside the limit of all available primitive data types.

For example factorial of 100 contains 158 digits in it so we can't store it in any primitive data type available. We can store as large Integer as we want in it. There is no theoretical limit on the upper bound of the range because memory is allocated dynamically but practically as memory is limited you can store a number which has Integer.MAX_VALUE number of bits in it which should be sufficient to store mostly all large values.

Below is an example Java program that uses BigInteger to compute Factorial.

```
// Java program to find large factorials using BigInteger
import java.math.BigInteger;
import java.util.Scanner;

public class Example
{
    // Returns Factorial of N
    static BigInteger factorial(int N)
    {
        // Initialize result
        BigInteger f = new BigInteger("1"); // Or BigInteger.ONE

        // Multiply f with 2, 3, ...N
        for (int i = 2; i <= N; i++)
            f = f.multiply(BigInteger.valueOf(i));

        return f;
    }

    // Driver method
    public static void main(String args[]) throws Exception
    {
        int N = 20;
        System.out.println(factorial(N));
    }
}
```

Output:

```
2432902008176640000
```

If we have to write above program in C++, that would be too large and complex, we can look at Factorial of Large Number.
In this way BigInteger class is very handy to use because of its large method library and it is also used a lot in competitive programming.
Now below is given a list of simple statements in primitive arithmetic and its analogous statement in terms of BigInteger objects.

Declaration

```
int a, b;

BigInteger A, B;
```

Initialization:

```
a = 54;
b = 23;
A  = BigInteger.valueOf(54);
B  = BigInteger.valueOf(37);
```

And for Integers available as string you can initialize them as:

```
A  = new BigInteger(“54”);
B  = new BigInteger(“123456789123456789”);
```

Some constant are also defined in BigInteger class for ease of initialization :

```
A = BigInteger.ONE;
// Other than this, available constant are BigInteger.ZERO
// and BigInteger.TEN
```

Mathematical operations:

```
int c = a + b;
BigInteger C = A.add(B);
```

Other similar function are subtract() , multiply(), divide(), remainder()
But all these function take BigInteger as their argument so if we want these 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:

```
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
```

Methods of BigInteger Class:

1. **BigInteger abs()**: This method returns a BigInteger whose value is the absolute value of this BigInteger.
2. **BigInteger add(BigInteger val)**: This method returns a BigInteger whose value is (this + val).
3. **BigInteger and(BigInteger val)**: This method returns a BigInteger whose value is (this & val).
4. **BigInteger andNot(BigInteger val)**: This method returns a BigInteger whose value is (this & ~val).
5. **int bitCount()**: This method returns the number of bits in the two's complement representation of this BigInteger that differ from its sign bit.
6. **int bitLength()**: This method returns the number of bits in the minimal two's-complement representation of this BigInteger, excluding a sign bit.
7. **byte byteValueExact()**: This method converts this BigInteger to a byte, checking for lost information.
8. **BigInteger clearBit(int n)**: This method returns a BigInteger whose value is equivalent to this BigInteger with the designated bit cleared.
9. **int compareTo(BigInteger val)**: This method compares this BigInteger with the specified BigInteger.
10. **BigInteger divide(BigInteger val)**: This method returns a BigInteger whose value is (this / val).
11. **BigInteger[] divideAndRemainder(BigInteger val)**: This method returns an array of two BigIntegers containing (this / val) followed by (this % val).
12. **double doubleValue()**: This method converts this BigInteger to a double.
13. **boolean equals(Object x)**: This method compares this BigInteger with the specified Object for equality.
14. **BigInteger flipBit(int n)**: This method returns a BigInteger whose value is equivalent to this BigInteger with the designated bit flipped.
15. **float floatValue()**: This method converts this BigInteger to a float.
16. **BigInteger gcd(BigInteger val)**: This method returns a BigInteger whose value is the greatest common divisor of abs(this) and abs(val).
17. **int getLowestSetBit()**: This method 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).
18. **int hashCode()**: This method returns the hash code for this BigInteger.
19. **int intValue()**: This method converts this BigInteger to an int.
20. **int intValueExact()**: This method converts this BigInteger to an int, checking for lost information.
21. **boolean isProbablePrime(int certainty)**: This method returns true if this BigInteger is probably prime, false if it's definitely composite.
22. **long longValue()**: This method converts this BigInteger to a long.
23. **long longValueExact()**: This method converts this BigInteger to a long, checking for lost information.
24. **BigInteger max(BigInteger val)**: This method returns the maximum of this BigInteger and val.
25. **BigInteger min(BigInteger val)**: This method returns the minimum of this BigInteger and val.

26. **BigInteger mod(BigInteger m)**: This method returns a BigInteger whose value is (this mod m).
27. **BigInteger modInverse(BigInteger m)**: This method returns a BigInteger whose value is (this-1 mod m).
28. **BigInteger modPow(BigInteger exponent, BigInteger m)**: This method returns a BigInteger whose value is (thisexponent mod m).
29. **BigInteger multiply(BigInteger val)**: This method returns a BigInteger whose value is (this * val).
30. **BigInteger negate()**: This method returns a BigInteger whose value is (-this).
31. **BigInteger nextProbablePrime()**: This method returns the first integer greater than this BigInteger that is probably prime.
32. **BigInteger not()**: This method returns a BigInteger whose value is (~this).
33. **BigInteger or(BigInteger val)**: This method returns a BigInteger whose value is (this | val).
34. **BigInteger pow(int exponent)**: This method returns a BigInteger whose value is (thisexponent).
35. **static BigInteger probablePrime(int bitLength, Random rnd)**: This method returns a positive BigInteger that is probably prime, with the specified bitLength.
36. **BigInteger remainder(BigInteger val)**: This method returns a BigInteger whose value is (this % val).
37. **BigInteger setBit(int n)**: This method returns a BigInteger whose value is equivalent to this BigInteger with the designated bit set.
38. **BigInteger shiftLeft(int n)**: This method returns a BigInteger whose value is (this << n).
39. **BigInteger shiftRight(int n)**: This method returns a BigInteger whose value is (this >> n).
40. **short shortValueExact()**: This method converts this BigInteger to a short, checking for lost information.
41. **int signum()**: This method returns the signum function of this BigInteger.
42. **BigInteger sqrt()**: This method returns the integer square root of this BigInteger.
43. **BigInteger[] sqrtAndRemainder()**: This method returns an array of two BigIntegers containing the integer square root s of this and its remainder this – s*s, respectively.
44. **BigInteger subtract(BigInteger val)**: This method returns a BigInteger whose value is (this – val).
45. **boolean testBit(int n)**: This method returns true if and only if the designated bit is set.
46. **byte[] toByteArray()**: This method returns a byte array containing the two's-complement representation of this BigInteger.
47. **String toString()**: This method returns the decimal String representation of this BigInteger.
48. **String toString(int radix)**: This method returns the String representation of this BigInteger in the given radix.
49. **static BigInteger valueOf(long val)**: This method returns a BigInteger whose value is equal to that of the specified long.
50. **BigInteger xor(BigInteger val)**: This method returns a BigInteger whose value is (this ^ val).