

বিগ-ইন্টিজার (জাভা)

জাভার একটি চমৎকার জিনিস হচ্ছে বিগ ইন্টিজার (BigInteger)। সেটি নিয়ে একাধিক পর্বের সিরিজ লিখেছে **তুষার রায়** (বাংলাদেশ ইউনিভার্সিটি অব বিজনেস এন্ড টেকনোলজি (বি ইউ বি টি)-তে কম্পিউটার বিজ্ঞান বিভাগে অধ্যয়নরত)।

আমরা বিভিন্ন প্রোগ্রামিং ভাষায় হিসাব-নিকাশ করার জন্য সাধারণত int (32-bit) বা long (64-bit) টাইপের ডাটা ব্যবহার করে থাকি। এদের মধ্যে int ডাটা দিয়ে খুব বড় সংখ্যাগুলোর হিসাব-নিকাশ করা না গেলেও long ডাটা দিয়ে মোটামুটি 10^{18} এর কাছাকাছি সংখ্যাগুলো এক্সেস করা যায়। সমস্যা হয় তখন যখন আমাদেরকে এর চেয়েও অনেক অনেক বড় সংখ্যা নিয়ে কাজ করার দরকার হয়। তখন আমরা কোনো সাধারণ ডাটা টাইপ দিয়ে সরাসরি এদের হিসাব-নিকাশ করতে পারি না, কারণ এতে করে আমাদের ভেরিয়েবলগুলোতে ডাটার ওভারফ্লো হয়।

এ ধরনের সমস্যা দূর করার জন্য Java-তে BigInteger নামের একটি বিল্ট-ইন ক্লাস তৈরি করা আছে। যা দিয়ে খুব সহজেই অনেক বড় বড় সংখ্যার হিসাব-নিকাশ করে ফেলা যায়। BigInteger-এর সাইজ JVM-এর পর্যাপ্ত মেমোরির উপর নির্ভর করে। তবে প্রয়োজন ছাড়া এটি ব্যবহার করা উচিত নয়, কারণ BigInteger সাধারণ ডাটা-টাইপ এর তুলনায় অনেক ধীরে কাজ করে। এখন তাহলে দেখা যাক, বিগ ইন্টিজার ব্যবহার করে আমরা কী কী কাজ করতে পারি এবং কীভাবে।

প্যাকেজ ইম্পোর্ট করা:

BigInteger ক্লাসটি জাভার math প্যাকেজের অন্তর্ভুক্ত। তাই BigInteger ব্যবহার করার জন্য আমাদেরকে প্রথমেই math প্যাকেজটি ইম্পোর্ট করে নিতে হবে।

```
import java.math.BigInteger;
```

ভেরিয়েবল ডিক্লেয়ার করা:

বিগ ইন্টিজার ভেরিয়েবল ডিক্লেয়ার করা অন্য সব ডাটা টাইপের মতোই। এখানে মূলত BigInteger ক্লাসের অবজেক্ট তৈরি করা হয়। যেমন, আমরা যদি a, b এবং c নামের তিনটি ভেরিয়েবল ডিক্লেয়ার করতে চাই তাহলে আমাদের এভাবে লিখতে হবে -

```
BigInteger a, b, c;
```

ইনপুট ও আউটপুট:

BigInteger এর ইনপুট ও আউটপুটও অন্য সব ডাটা টাইপের মতো। যেমন, আমরা যদি একটি ভেরিয়েবলের মান ইনপুট নিয়ে তা আউটপুটে দেখাতে চাই তাহলে আমাদের প্রোগ্রাম টি এভাবে লিখতে হবে -

```
import java.util.*;
import java.math.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        BigInteger n;
        n = sc.nextBigInteger();
        System.out.println(n);
    }
}
```

ভ্যালু অ্যাসাইন করা:

আমাদেরকে অনেক সময় এক ভেরিয়েবলের মান অন্য ভেরিয়েবলে অথবা সরাসরি কোন মান একটি ভেরিয়েবলে অ্যাসাইন করার প্রয়োজন হতে পারে। এক ভেরিয়েবল থেকে অন্য ভেরিয়েবলে ভ্যালু অ্যাসাইন করতে আমরা সরাসরি '=' ব্যবহার করতে পারি। যেমন, একটি ভেরিয়েবল b-এর মান অপর একটি ভেরিয়েবল a-তে অ্যাসাইন করতে চাইলে হলে লিখতে হবে -

```
BigInteger a = b;
```

কিন্তু সরাসরি যদি কোনো মান একটি ভেরিয়েবলে অ্যাসাইন করতে চাই, তাহলে BigInteger এর কন্সট্রাক্টর ব্যবহার করে ভেরিয়েবলে (যেটি আসলে একটি BigInteger অবজেক্ট) তার মান অ্যাসাইন করতে হবে।

BigInteger-এর কন্সট্রাক্টরটি প্যারামিটার হিসেবে একটি নাম্বারের স্ট্রিং নিয়ে থাকে। যেমন -

```
BigInteger n = new BigInteger("1234567890123456789");
```

আবার আমরা BigInteger.valueOf(int_value) ব্যবহার করে যেকোনো ইন্টিজারকে সরাসরি BigInteger হিসেবে ব্যবহার করতে পারি।

BigInteger অ্যারে:

BigInteger এর আবার অ্যারে! অবাক হবার কিছু নেই। বিগ-ইন্টিজারের অ্যারেও আমরা ব্যবহার করতে পারি। এটিও অন্য সব ডাটা টাইপের অ্যারের মতোই।

```
BigInteger[] array_name = new BigInteger[array_size];
```

এখন আমরা চাইলে এই অ্যারেটি সর্টও করে ফেলতে পারি। সম্পূর্ণ অ্যারে সর্ট করতে চাইলে লিখতে হবে Arrays.sort(array_name);, আর কোনো একটা নির্দিষ্ট রেঞ্জের ভ্যালু গুলো সর্ট করতে চাইলে লিখতে হবে Arrays.sort(array_name, fromIndex, toIndex+1);। সর্ট করার ক্ষেত্রে খেয়াল রাখতে হবে যাতে অ্যারেটির কোনো ইন্ডেক্সে নাল ভ্যালু না থাকে, নইলে আমাদের প্রোগ্রামটি এক্সেপশন থ্রো করবে।

BigInteger এর মেথড

জাভার BigInteger ক্লাসে অনেক গুলো মেথড আছে, যা ব্যবহার করে আমরা বিগ-ইন্টিজার দিয়ে অনেক বড় বড় সমস্যা খুব সহজেই সমাধান করে ফেলতে পারি। আমি এখানে প্রয়োজনীয় কিছু মেথড নিয়ে আলচনা করছি।

যোগ / বিয়োগ / গুণ / ভাগ করা

দুটি বিগ-ইন্টিজার যোগ করার জন্য আমাদেরকে add() মেথডটি ব্যবহার করতে হবে। ধরা যাক, আমরা a এবং b দুটি সংখ্যা যোগ করে ans নামের একটি ভেরিয়েবলে রাখতে চাই। তাহলে আমাদের লিখতে হবে -

```
BigInteger ans = a.add(b);
```

একই ভাবে বিয়োগ করার জন্য subtract(), গুণ করার জন্য multiply(), এবং ভাগ করার জন্য আমাদেরকে divide() মেথড ব্যবহার করতে হবে।

ভাগশেষ বের করা

একটি সংখ্যা কে অপর একটি সংখ্যা দিয়ে ভাগ করার পর অবশিষ্ট ভাগশেষ বা mod ভ্যালু বের করতে চাইলে আমাদেরকে mod() মেথড টি ব্যবহার করতে হবে। যেমন, আমরা যদি একটি সংখ্যা a কে অপর একটি সংখ্যা m দিয়ে mod করতে চাই তাহলে আমাদের লিখতে হবে-

```
BigInteger ans = a.mod(m);
```

এছাড়াও remainder() মেথড দিয়েও একই কাজ করা যায়।

তুলনা করা

অন্য সব ডাটা টাইপের মত বিগ ইন্টিজারে আমরা দুটি সংখ্যা তুলনা করার জন্য ==, > বা < ব্যবহার করতে পারি না। দুটি বিগ-ইন্টিজার সমান কিনা তা দেখার জন্য আমাদের equals() মেথড টি ব্যবহার করতে হয়। সংখ্যা দুটি সমান হলে মেথডটি true রিটার্ন করে, তা নাহলে false রিটার্ন করে। দুটি বিগ-ইন্টিজার কম্পেয়ার করার জন্য জাভার BigInteger ক্লাসে compareTo() মেথড আছে। এই মেথডটি একটি ইন্টিজার সংখ্যা রিটার্ন করে। আমরা যখন দুটি সংখ্যা কম্পেয়ার করবো তখন প্রথম সংখ্যাটি যদি দ্বিতীয় সংখ্যার চেয়ে ছোটো হয় তাহলে মেথডটি -1 রিটার্ন করবে, সংখ্যা দুটি যদি সমান হয় তাহলে 0 রিটার্ন করবে এবং প্রথম সংখ্যাটি যদি বড় হয় তাহলে 1 রিটার্ন করবে। নিচের কোডটি রান করে দেখলে বিষয়টি আরো ভালভাবে বুঝা যাবে -

```

import java.util.*;
import java.math.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        BigInteger a, b;
        a = sc.nextBigInteger();
        b = sc.nextBigInteger();
        int cmp = a.compareTo(b);
        if(cmp == -1)
            System.out.println("a is less than b");
        else if(cmp == 0)
            System.out.println("a is equal to b");
        else if(cmp == 1)
            System.out.println("a is greater than b");
    }
}

```

পাওয়ার বের করা

আমরা চাইলে বিগ-ইন্টিজারের পাওয়ারও বের করতে পারি। এজন্য আমাদেরকে pow() মেথডটি ব্যবহার করতে হবে। তবে এই মেথডটি প্যারামিটার হিসেবে একটি ইন্টিজার গ্রহন করে, তাই পাওয়ার এর মানটি অবশ্যই int ডাটা টাইপের রেঞ্জের মধ্যে হতে হবে। কোনো একটি সংখ্যা n এর পাওয়ার p বের করতে চাইলে আমাদের লিখতে হবে -

```
BigInteger ans = n.pow(p);
```

বিগ-মড ভ্যালু বের করা

বিগ-মড বা $(n^p) \% m$ এক্সপ্লেসনের মান বের করার জন্য বিগ-ইন্টিজারের modpow() মেথড আছে। মেথডটি প্যারামিটার হিসেবে দুটি BigInteger নিয়ে থাকে। এখন আমরা যদি কোন একটি সংখ্যা n-এর পাওয়ার p-কে অপর একটি সংখ্যা m দিয়ে mod করতে চাই, তাহলে আমাদের লিখতে হবে -

```
BigInteger ans = n.modPow(p,m);
```

মডুলার-ইনভার্স ভ্যালু বের করা

মডুলার-ইনভার্স বের করার জন্য বিগ-ইন্টিজারে modInverse() মেথড আছে। তবে এজন্য আমরা যে সংখ্যার মডুলার-ইনভার্স বের করতে চাই এবং যে সংখ্যাটি দিয়ে mod করতে চাই, তাদেরকে অবশ্যই রিলেটিভলি-প্রাইম হতে হবে। এখন আমরা যদি কোনো একটি সংখ্যা n-এর মডুলার-ইনভার্স বের করতে চাই এবং যে সংখ্যাটি দিয়ে mod করবো তা যদি m হয়, তাহলে আমাদের লিখতে হবে -

```
BigInteger ans = n.modInverse(m);
```

GCD বের করা

দুটি সংখ্যার Greatest Common Divisor (GCD) অর্থাৎ গসাগু বের করার জন্য বিগ-ইন্টিজারে gcd() মেথডটি আছে। দুটি সংখ্যা a এবং b-এর GCD বের করতে চাইলে আমাদের লিখতে হবে -

```
BigInteger ans = a.gcd(b);
```

ম্যাক্সিমাম/ মিনিমাম বের করা

জাভার বিগ-ইন্টিজারে দুটি সংখ্যার মধ্যে বৃহত্তর সংখ্যা বের করার জন্য max() এবং ক্ষুদ্রতর সংখ্যা বের করার জন্য min() মেথড রয়েছে। দুটি সংখ্যার মধ্যে ম্যাক্সিমাম বা মিনিমাম বের করার জন্য আমাদের লিখতে হবে -

```
BigInteger largest = a.max(b);
```

```
BigInteger smallest = a.min(b);
```

Absolute মান বের করা

কোনো সংখ্যার পরম মান বা Absolute value বের করার জন্য বিগ-ইন্টিজারে abs() মেথড আছে। কোনো একটি সংখ্যা n এর Absolute মান বের করতে চাইলে লিখতে হবে -

```
BigInteger ans = n.abs();
```

সম্ভাব্য-মৌলিক সংখ্যা

আমাদের অনেক সময় প্রাইম নাম্বার বের করার প্রয়োজন হতে পারে। বিশেষ করে ক্রিপ্টোগ্রাফীর জন্য প্রাইম নাম্বার জেনারেট করা অনেক জরুরি। একটি বিগ-ইন্টিজার সম্ভাব্য প্রাইম নাম্বার কি না তাও আমরা বের করতে পারি। এজন্য BigInteger ক্লাসে isProbablePrime() মেথড আছে। মেথডটিতে probabilistic primality টেস্টের জন্য Miller-Rabin অ্যালগরিদম ব্যবহার করা হয়েছে। মেথডটি প্যারামিটারে certainty হিসেবে একটি ইন্টিজার ভ্যালু নিয়ে থাকে এবং একটি বুলিয়ান ডাটা রিটার্ন করে। certainty দিয়ে অসম্ভাব্যতার হার বের করা হয়। তারপর তা থেকে সম্ভাব্যতার হার বের করা হয়। সম্ভাব্যতার হার বের করা হয় $(1 - 0.5^{\text{certainty}})$ এভাবে। কোনো একটি সংখ্যা প্রাইম নাম্বার হবার সম্ভাব্যতা যদি এই মানকে অতিক্রম করে তাহলে মেথড টি true রিটার্ন করে, অন্যথায় false রিটার্ন করে। অর্থাৎ certainty এর মান যত বড় হবে, সম্ভাব্যতার হার তত বাড়বে। আবার অন্য দিকে সম্ভাব্যতার হার যত বাড়বে, প্রোগ্রামের এক্সিকিউশন টাইম ততো বেশি লাগবে। তাই এটা ব্যবহারকারীর উপর নির্ভর করে যে সে কতটুকু সময়ের মধ্যে কত সম্ভাব্যতার প্রাইম নাম্বার চায়।

কোন একটি সংখ্যা n, certainty c-এর সাপেক্ষে প্রাইম কি না, বের করতে হলে আমাদের লিখতে হবে -

```
boolean flag = n.isProbablePrime(c);
```

আবার কোনো একটি সংখ্যার পরবর্তী কোন সংখ্যাটি একটি সম্ভাব্য প্রাইম, তা বের করার জন্য বিগ ইন্টিজারে আছে nextProbablePrime() মেথড। মেথডটি একটি সম্ভাব্য প্রাইম নাম্বার রিটার্ন করে এবং রিটার্ন করা নাম্বারটি একটি কম্পোজিট নাম্বার হবার সম্ভাব্যতা থাকে 2-100 এর কম। কোনো একটি সংখ্যা n-এর পরবর্তী সম্ভাব্য প্রাইম সংখ্যাটি বের করতে হলে আমাদের লিখতে হবে-

```
BigInteger ans = n.nextProbablePrime();
```

আমাদের অনেক সময় র্যান্ডম প্রাইম নাম্বারের দরকার হতে পারে। বিগ ইন্টিজারে র্যান্ডম ভাবেও সম্ভাব্য প্রাইম নাম্বার জেনারেট করা যায়। এজন্য বিগ ইন্টিজারে `probablePrime()` মেথড আছে। মেথডটি প্যারামিটারে `bitLength` হিসেবে একটি ইন্টিজার এবং `rnd` হিসেবে একটি র্যান্ডম ডাটা নিয়ে থাকে। মেথডটির রিটার্ন করা নাম্বারটি এই `bitLength`-সংখ্যক বিটের হবে। মেথডটি কল করার সময় এর মান সব সময় 2 বা তার বেশি হতে হবে। আর `rnd` হচ্ছে একটি র্যান্ডম বিট সোর্স, একটি সম্ভাব্য প্রাইম নাম্বার রিটার্ন করার জন্য যার primality টেস্ট করা হয়। মেথডটি একটি সম্ভাব্য প্রাইম নাম্বার রিটার্ন করে এবং রিটার্ন করা নাম্বারটি একটি কম্পোজিট নাম্বার হবার সম্ভাব্যতা থাকে 2^{-100} এর কম। এখন আমাদের যদি `n` সংখ্যক বিটের একটি র্যান্ডম সম্ভাব্য প্রাইম নাম্বার বের করতে হয়, তাহলে আমাদের লিখতে হবে -

```
Random rnd = new Random();  
BigInteger ans = BigInteger.probablePrime(n, rnd);
```

Bitwise অপারেশন

বিগ ইন্টিজারেও আমরা বিট-ওয়াইজ অপারেশন করতে পারি। এজন্য `BigInteger` ক্লাসে কিছু বিল্ট-ইন মেথড রয়েছে।

AND (`a&b`): বিগ ইন্টিজারের বিট-ওয়াইজ AND ভ্যালু বের করার জন্য রয়েছে `and()` মেথড। দুটি বিগ ইন্টিজার `a` এবং `b`-এর AND ভ্যালু বের করতে চাইলে লিখতে হবে -

```
BigInteger ans = a.and(b);
```

OR (`a|b`): বিগ ইন্টিজারের বিট-ওয়াইজ OR ভ্যালু বের করার জন্য রয়েছে `or()` মেথড। দুটি বিগ ইন্টিজার `a` এবং `b`-এর OR ভ্যালু বের করতে চাইলে আমাদের লিখতে হবে -

```
BigInteger ans = a.or(b);
```

XOR (`a^b`): বিগ ইন্টিজারে `xor()` মেথড টি দিয়ে বিট-ওয়াইজ XOR বা Exclusive OR ভ্যালু বের করা হয়। দুটি বিগ ইন্টিজার `a` এবং `b` এর XOR ভ্যালু বের করতে চাইলে আমাদের লিখতে হবে-

```
BigInteger ans = a.xor(b);
```

NOT (`~a`): বিগ ইন্টিজারের বিট-ওয়াইজ NOT ভ্যালু বের করার জন্য রয়েছে `not()` মেথড। কোন একটি বিগ ইন্টিজার `a` এর NOT ভ্যালু বের করতে চাইলে আমাদের লিখতে হবে-

```
BigInteger ans = a.not();
```

Shift-Left (`a<<n`): আমরা চাইলে বিগ ইন্টিজারের বিটগুলোকে শিফটও করতে পারি। বিটগুলোকে বাম দিকে শিফট করার জন্য আমাদেরকে `shiftLeft()` মেথডটি ব্যবহার করতে হবে। মেথডটি প্যারামিটারে একটি ইন্টিজার সংখ্যা নিয়ে থাকে। একটি নাম্বারের বিটগুলোকে আমরা বাম দিকে কত বিট শিফট করবো এটি হচ্ছে তার মান। কোনো একটি সংখ্যা `a`-কে আমরা যদি `n` বিট বামে শিফট করতে চাই, তাহলে লিখতে হবে -

```
BigInteger ans = a.shiftLeft(n);
```

Shift-Right ($a \gg n$): বিগ ইন্টিজারের বিট গুলো কে ডান দিকে শিফট করার জন্য আমাদেরকে `shiftRight()` মেথডটি ব্যবহার করতে হবে। এটি `shiftLeft()` মেথডটির মতই। কোনো একটি সংখ্যা a -কে আমরা যদি n বিট ডানে শিফট করতে চাই, তাহলে লিখতে হবে -

```
BigInteger ans = a.shiftRight(n);
```

Test-Bit ($a \& (1 \ll n)$): একটি বিগ ইন্টিজারের n -তম বিটটি 0 নাকি 1, তা বের করার জন্য `BigInteger` ক্লাসে আছে `testBit()` মেথড। মেথডটি প্যারামিটারে n -এর মান নেয় এবং n -তম বিট 1 হলে `true` রিটার্ন করে, আর 0 হলে `false` রিটার্ন করে। এখন আমরা যদি কোনো একটি সংখ্যা a -এর n -তম বিট পরীক্ষা করতে চাই, তাহলে লিখতে হবে-

```
boolean flag = a.testBit(n);
```

Clear-Bit ($a \& \sim(1 \ll n)$): একটি বিগ ইন্টিজারের কোনো একটি বিটকে 0 করতে হলে আমাদের `clearBit()` মেথডটি ব্যবহার করতে হবে। কোনো একটি সংখ্যা a -এর n -তম বিটকে 0 করতে চাইলে আমাদের লিখতে হবে-

```
BigInteger ans = a.clearBit(n);
```

Set-Bit ($a \mid (1 \ll n)$): একটি বিগ ইন্টিজারের কোনো একটি বিটকে 1 করতে হলে আমাদের `setBit()` মেথডটি ব্যবহার করতে হবে। কোনো একটি সংখ্যা a -এর n -তম বিটকে 1 করতে চাইলে আমাদের লিখতে হবে -

```
BigInteger ans = a.setBit(n);
```

Flip-Bit ($a \wedge (1 \ll n)$): একটি বিগ ইন্টিজারের কোনো একটি বিটকে ফ্লিপ করতে হলে অর্থাৎ 0 থাকলে 1 করতে অথবা 1 থাকলে 0 করতে চাইলে আমাদের `flipBit()` মেথডটি ব্যবহার করতে হবে। কোনো একটি সংখ্যা a -এর n -তম বিটকে ফ্লিপ করতে চাইলে আমাদের লিখতে হবে-

```
BigInteger ans = a.flipBit(n);
```

AND-NOT ($a \& \sim b$): দুটি বিগ ইন্টিজারের বিট-ওয়াইজ AND-NOT ভ্যালু বের করতে চাইলে `andNot()` মেথডটি ব্যবহার করতে হবে। দুটি বিগ ইন্টিজার a এবং b এর AND-NOT ভ্যালু বের করতে চাইলে আমাদের লিখতে হবে-

```
BigInteger ans = a.andNot(b);
```

Lowest Set Bit ($\log_2(a \& -a)$): কোনো সংখ্যার সবচেয়ে ছোট কোন বিটে 1 আছে, তা বের করার জন্য বিগ ইন্টিজারে `getLowestSetBit()` মেথড আছে। কোনো একটি বিগ ইন্টিজার a -এর সবচেয়ে ছোট কোন বিটটি 1 তা বের করতে চাইলে আমাদের লিখতে হবে -

```
int ans = a.getLowestSetBit();
```

আরো কিছু প্রয়োজনীয় মেথড

`toString()`: বিগ ইন্টিজার কে স্ট্রিং এ কনভার্ট করতে ব্যবহার করা হয়।

`toArray()`: বিগ ইন্টিজার কে বাইট-অ্যারে তে কনভার্ট করতে ব্যবহার করা হয়।

`negate()`: বিগ ইন্টিজারের সাইন পরিবর্তন করতে ব্যবহার করা হয়।

`bitCount()`: কোন সংখ্যার 2's complement এ সাইন বিটের বিপরীত কয়টি বিট আছে তা বের করার জন্য ব্যবহার করা হয়।

`bitLength()`: একটি সংখ্যা কে বাইনারি বৈজ্ঞেয় রিপ্রেজেন্ট করতে (সাইন বিট ছাড়া) কয়টি বিটের প্রয়োজন তা জানার জন্য ব্যবহার করা হয়।

`signum()`: কোন সংখ্যা ধনাত্মক, ঋণাত্মক অথবা শূন্য কি না তা জানার জন্য ব্যবহার করা হয়। ধনাত্মক হলে 1, ঋণাত্মক হলে -1 এবং শূন্য হলে মেথড টি 0 রিটার্ন করে।

`intValue()`: `BigInteger` থেকে `int` ডাটায় কনভার্ট করতে ব্যবহার করা হয়। তবে এ ক্ষেত্রে ডাটা ওভারফ্লো হলে মেথড টি কোন এক্সেপশন থ্রো করে না। এক্ষেত্রে `intValueExact()` ব্যবহার করলে এই মেথড টি এক্সেপশন থ্রো করে।

`longValue()`: `BigInteger` থেকে `long` ডাটায় কনভার্ট করতে ব্যবহার করা হয়। এই মেথড টি ও ডাটা ওভারফ্লো হলে এক্সেপশন থ্রো করে না। এক্ষেত্রে `longValueExact()` ব্যবহার করলে এই মেথড টি এক্সেপশন থ্রো করে।

`floatValue()`: `BigInteger` থেকে `float` ডাটায় কনভার্ট করতে ব্যবহার করা হয়।

`doubleValue()`: `BigInteger` থেকে `double` ডাটায় কনভার্ট করতে ব্যবহার করা হয়।

`hashCode()`: হ্যাশ-কোড বের করার জন্য ব্যবহার করা হয়।

`divideAndRemainder()`: মেথড টি দুটি সংখ্যার ভাগফল এবং ভাগশেষ এক সাথে একটি `BigInteger` অ্যারে আকারে রিটার্ন করে। রিটার্ন করা অ্যারে টির [0] ইন্ডেক্সে থাকে ভাগফলের মান এবং [1] ইন্ডেক্সে থাকে ভাগশেষ এর মান।


```
import java.util.*;
import java.math.*;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        BigInteger a=sc.nextBigInteger();
        BigInteger b=sc.nextBigInteger();
        BigInteger arr[] = a.divideAndRemainder(b);
        System.out.println("Quotient = "+arr[0]);
        System.out.println("Remainder = "+arr[1]);
    }
}
```

এই ছিলো আমাদের BigInteger নিয়ে আলোচনা। Java-তে BigDecimal নামে আরো একটি ক্লাস আছে যা দিয়ে Decimal সংখ্যার হিসাব নিকাশ করা যায়। এটির ব্যবহার BigInteger-এর মতোই।

Some Problems:

- [Codeforces 66A - Petya and Java](#)
- [Light OJ 1214 - Large Division](#)
- [UVA 424 - Integer Inquiry](#)
- [UVA 465 - Overflow](#)
- [UVA 495 - Fibonacci Freeze](#)
- [UVA 619 - Numerically Speaking](#)
- [UVA 623 - 500!](#)
- [UVA 713 - Adding Reversed Numbers](#)
- [UVA 748 - Exponentiation](#)
- [UVA 1226 - Numerical surprises](#)
- [UVA 10013 - Super long sums](#)
- [UVA 10083 - Division](#)
- [UVA 10106 - Product](#)
- [UVA 10464 - Big Big Real Numbers](#)
- [UVA 10494 - If We Were a Child Again](#)
- [UVA 10523 - Very Easy !!!](#)
- [UVA 10551 - Basic Remains](#)
- [UVA 10579 - Fibonacci Numbers](#)
- [UVA 10814 - Simplifying Fractions](#)
- [UVA 10925 - Krakovia](#)
- [UVA 10929 - You can say 11](#)
- [UVA 11448 - Who said crisis?](#)
- [UVA 11879 - Multiple of 17](#)