Theory: BigInteger

21 minutes 0 / 5 problems solved

§1. Using large numbers in Java

As you might remember, the standard primitive integer types cannot store very large numbers. It is not possible to assign the following large number to a variable of the type `int` (or even `long`):

```
int y = 629572917952287634062530 98; // compilation-
error: integer number too large
```

That is also the reason why operations with numbers can sometimes lead to **type overflow**. For example, check out the following code:

```
int a = Integer.MAX_VALUE; // 2147483647
a += 2; // -2147483647
```

Fortunately, the Java Class Library provides a class called `BigInteger` for processing very large numbers (both positive and negative). The size of a stored number is only limited by the available memory.

The `BigInteger` class is **immutable** which means methods of the class return new instances instead of changing existing ones.

Although this type can store any integers, including small numbers, `BigInteger` should only be used if it is absolutely necessary. Using it is less intuitive compared to built-in types and, more importantly, there is always a performance hit associated with its use. `BigInteger` operations are substantially slower than operations on built-in integer types.

§2. Creating objects of BigInteger

The class `BigInteger` belongs to the `java.math` package. We import it by writing the following statement:

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

Here is an instance of the class that stores the large number presented above:

```
BigInteger number = new BigInteger("629572917952287634062530 98");
```

It is also possible to create an instance by passing a long value to the method `valueOf`:

```
BigInteger number = BigInteger.valueOf(1_000_000_000);
```

In addition, the class has several useful constants:

```
BigInteger zero = BigInteger.ZERO; // 0
BigInteger one = BigInteger.ONE;   // 1
BigInteger ten = BigInteger.TEN;   // 10
```

Using them is a good practice because constants allow you to reuse an already created object. This is particularly important considering that instance of `BigInteger` is actually quite big. Except for a memory optimization point, the code with constants is easier to read.

Compare the code below:

```
if (something) {
    return new BigInteger("1");
}
```

Current topic:

BigInteger

Topic depends on:

Stage 3
✓ Objects

Table of contents:

↑ BigInteger

§1. Using large numbers in Java

§2. Creating objects of BigInteger

§3. Methods of BigInteger

Feedback & Comments

and its analog with constants:

```
if (something) {
    return BigInteger.ONE;
}
```

Remember it and try to use built-In `BigInteger` constants whenever possible.

§3. Methods of BigInteger

The class has a set of non-static methods to perform all standard arithmetic operations. The following example demonstrates the addition.

```
BigInteger eleven = ten.add(one);
System.out.println(eleven); // 11

System.out.println(ten); // 10, it has not changed!
```

Keep in mind, that the arithmetic methods do not change instances but create a new one.

Other arithmetic methods (subtraction, multiplication, integer division) are listed below:

```
BigInteger nine = ten.subtract(BigInteger.ONE); // 10 - 1 = 9
BigInteger oneHundredTen = ten.multiply(eleven); // 10 * 11 = 110
BigInteger twelve = oneHundredTen.divide(nine); // integer division: 12
```

The method `negate` returns a new `BigInteger` with the changed sign, like this:

```
nine.negate(); // -9
```

The method `divideAndRemainder` returns an array consisting of two numbers: the result of integer division and the remainder.

```
BigInteger[] pair = oneHundredTen.divideAndRemainder(nine); // 12 and 2
```

The class also provides methods for performing more complex math operations. The method `abs` returns a new `BigInteger` whose value is the absolute value of this `BigInteger`.

```
BigInteger number = new BigInteger("-8");
System.out.println(number.abs()); // 8
```

The method `gcd` returns the greatest common divisor of two numbers.

```
BigInteger three = BigInteger.valueOf(3);
BigInteger six = BigInteger.valueOf(6);
System.out.println(three.gcd(six)); // 3
```

The class has methods for performing bitwise and bitshift operations as well, but we do not consider them here.