33. Converting BigInteger into a primitive type

The `BigInteger` class is a very handy tool for representing immutable arbitrary-precision integers.

This class also contains methods (originating from `java.lang.Number`) that are useful for converting `BigInteger` into a primitive type such as `byte`, `long`, or `double`. However, these methods can produce unexpected results and confusion. For example, let's assume that we have `BigInteger` that wraps `Long.MAX_VALUE`:

```
BigInteger nr = BigInteger.valueOf(Long.MAX_VALUE);
```

Let's convert this `BigInteger` into a primitive long via the `BigInteger.longValue()` method:

```
long nrLong = nr.longValue();
```

So far, everything has worked as expected since the `Long.MAX_VALUE` is 9,223,372,036,854,775,807 and the `nrLong` primitive variable has exactly this value.

Now, let's try to convert this `BigInteger` class into a primitive `int` value via the `BigInteger.intValue()` method:

```
int nrInt = nr.intValue();
```

This time, the `nrInt` primitive variable will have a value of -1 (the same result will produce `shortValue()` and `byteValue()`). Conforming to the documentation, if the value of `BigInteger` is too big to fit in the specified primitive type, only the low-order *n* bits are returned (*n* depends on the specified primitive type). But if the code is not aware of this statement, then it will push values as -1 in further computations, which will lead to confusion.

However, starting with JDK 8, a new set of methods was added. These methods are dedicated to identifying the information that's lost during the conversion from `BigInteger` into the specified primitive type. If a piece of lost information is detected, `ArithmeticException` will be thrown. This way, the code signals that the conversion has encountered some issues and prevents this unpleasant situation.

These methods are `longValueExact()`, `intValueExact()`, `shortValueExact()`, and `byteValueExact()`:

```
long nrExactLong = nr.longValueExact(); // works as expected
int nrExactInt = nr.intValueExact();    // throws ArithmeticException
```

Notice that `intValueExact()` did not return -1 as `intValue()`. This time, the lost information that was caused by the attempt of converting the largest `long` value into `int` was signaled via an exception of the `ArithmeticException` type.