Например, необходимо хранить и использовать в программе очень большие числа, которые выходят за пределы лопустимых значений для типов long и double. В этом случае для работы с числовыми данными можно использовать два дополнительных типа из пакета **java.math** - **BigInteger** (для целочисленных данных) и **BigDecimal** (для чисел с плавающей точкой).

Стоит отметить, несмотря на то, что объекты BigInteger и BigDecimal представляют числа, мы не можем применять с ними стандартные арифметические операции. Все математические действия с данными объектами идут через их методы.

BigInteger add(BigInteger other): возвращает сумму двух чисел

BigInteger subtract(BigInteger other): возвращает разность двух чисел

BigInteger multiply(BigInteger other): возвращает произведение двух чисел

Прежде всего в том, ****у них теоретически нет максимального размера****. Теоретически, потому что не бывает компьютеров с бесконечным размером памяти.

Поэтому можно сказать, что размер чисел BigInteger и BigDecimal практически ничем не ограничен.

Для чего используются эти классы? Прежде всего, ****для вычислений с крайне высокими требованиями к точности****. Есть, к примеру, программы, в которых от точности вычислений может зависеть человеческая жизнь (ПО для самолетов и ракет или для медицинского оборудования). Поэтому, если даже 150-й разряд после запятой играет важную роль, BigDecimal — лучший выбор. Кроме того, довольно часто эти объекты применяются в мире финансов, где точность вычислений вплоть до самых мелких значений тоже крайне важна.

Объекты этих классов создаются вот так:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

BigInteger integer = **new** BigInteger("11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111");

System.out.println(integer);

BigDecimal decimal = **new** BigDecimal("123.444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444444");

System.out.println(decimal);

}}

Передача строки в качестве параметра — только один из возможных конструкторов. Здесь мы используем строки, потому что наши числа превышают максимальные значения long и double, а как-то ведь надо объяснить компилятору, какое именно число мы хотим получить :) Просто передать в конструктор число 111111111111111111111111111111111111111111

не выйдет: Java попытается «вместить» переданное число в один из примитивных типов данных, но ни в один из них оно не влезет.

Еще один важный момент, который необходимо помнить при работе с классами больших чисел — ****их объекты являются неизменяемыми (**Immutable**)****.

С принципом неизменяемости ты уже хорошо знаком на примере класса String и классов-оберток для примитивов (Integer, Long и другими).

**import** java.math.BigInteger;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

BigInteger integer = **new** BigInteger("11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111");

System.out.println(integer);

integer.add(BigInteger.valueOf(33333333));

System.out.println(integer);

}}

****Вывод в консоль:** 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111**

Наше число не изменилось, как и следовало ожидать. Чтобы операция сложения прошла успешно, необходимо создать новый объект и присвоить ему результат сложения.

**import** java.math.BigInteger;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

BigInteger integer = **new** BigInteger("11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111");

System.out.println(integer);

BigInteger result = integer.add(BigInteger.valueOf(33333333));

System.out.println(result);

}}

****Вывод в консоль:** 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111 11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111144444444** Вот, теперь все работает как надо :)

****Классы больших чисел не используют в своей работе операторы +-\*/, а предоставляют вместо этого набор методов.****

****методы для осуществления арифметических операций:**** add(), subtract(), multiply(), divide(). Используются для операций сложения, вычитания, умножения и деления соответственно.

****doubleValue()****, ****intValue()****, ****floatValue()****, ****longValue()**** и т.д. — используются для преобразования большого числа к примитивному типу Java. Будь осторожен при их использовании и не забывай про разницу во вместимости!

****min()**** и ****max()**** — позволяют найти минимальное и максимальное значение из двух переданных больших чисел.  
Обрати внимание: методы не являются статическими!

Ты можешь установить количество цифр после запятой для числа BigDecimal при помощи метода ****setScale()****.****Всего у** BigDecimal **существует 8 режимов округления.****

Но в случае с объектами классов BigDecimal использовать метод equals() для сравнения не рекомендуется. Причина этого в том, что метод ****BigDecimal.equals()**** двух чисел возвращает true только в случае, если ****2 числа имеют одинаковое значение и масштаб (scale)****:

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Double a = 1.5;

Double b = 1.50;

System.out.println(a.equals(b));

BigDecimal x = **new** BigDecimal("1.5");

BigDecimal y = **new** BigDecimal("1.50");

System.out.println(x.equals(y));

}}

****Вывод в консоль:** true false** Как видишь, числа 1.5 и 1.50 в случае с BigDecimal оказались неравны! Это произошло именно из-за специфики работы метода equals(), в классе BigDecimal. Для более корректного сравнения двух BigDecimal лучше использовать метод compareTo():

**import** java.math.BigDecimal;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

BigDecimal x = **new** BigDecimal("1.5");

BigDecimal y = **new** BigDecimal("1.50");

System.out.println(x.compareTo(y));

}}

****Вывод в консоль:** 0** Метод compareTo() вернул 0, что означает равенство 1.5 и 1.50.