To determine how many calls to the nextValue method we can make before causing a long-integer overflow, let's consider the following:

* The ArithmeticProgression class typically works with a starting value and a common difference (increment). In this case, the increment is given as 128.
* The long type in Java has a maximum value of Long.MAX\_VALUE, which is 263−12^{63} - 1263−1 (9,223,372,036,854,775,807).
* The progression starts at some initial value (let's assume it starts at 0), and each call to nextValue adds the increment (128) to the current value.

**Step-by-step breakdown:**

1. **Initial Value (start)**: Assume the initial value is 0 for simplicity.
2. **Increment (d)**: 128
3. **Formula for the nth value in the progression**:

current value=start+n×d\text{current value} = \text{start} + n \times dcurrent value=start+n×d

where nnn is the number of calls to nextValue.

1. We need to determine when the current value exceeds Long.MAX\_VALUE.

0+n×128≤9,223,372,036,854,775,8070 + n \times 128 \leq 9,223,372,036,854,775,8070+n×128≤9,223,372,036,854,775,807

Solving for nnn:

n≤9,223,372,036,854,775,807128n \leq \frac{9,223,372,036,854,775,807}{128}n≤1289,223,372,036,854,775,807​

Let's calculate the value of nnn.

You can make **72,057,594,037,927,935** calls to the nextValue method before causing a long-integer overflow. ​​

# Long.MAX\_VALUE in Java

long\_max\_value = 9223372036854775807

increment = 128

# Calculate the maximum number of calls to nextValue before overflow

max\_calls = long\_max\_value // increment

max\_calls