Solution to exercise 1

In this exercise, you want to take care of possible overflows in the computation of

$$\texttt{r := (a * randomNumbers[indexOfInteger] + c) \% modulus}$$

under the assumption

$$\texttt{Long.MAX\_VALUE} < a \cdot \texttt{modulus} + c < 2 \cdot \texttt{Long.MAX\_VALUE}.$$

If the operation `a * randomNumbers[indexOfInteger] + c` produces an overflow, i.e., if the result is bigger than `Long.MAX_VALUE`, the value produced in Java is negative (note that a multiple overflow is prevented by the assumption $a \cdot \texttt{modulus} + c < 2 \cdot \texttt{Long.MAX\_VALUE}$).
In this case, two values play an essential role: the true mathematical value of

$$\texttt{a * randomNumbers[indexOfInteger] + c}$$

and the number you get in your program. The first one can be written as

$$\texttt{trueNumber} = \texttt{Long.MAX\_VALUE} + \texttt{valueOverflow}$$

where `valueOverflow` is the size of the overflow got in the operation, whereas the number produced by Java is

$$\texttt{observedNumber} = \texttt{Long.MIN\_VALUE} + \texttt{valueOverflow} - 1. \tag{1}$$

Note that, if one does not take the overflow into account, Java would simply return

$$\texttt{observedNumber \% modulus},$$

which might differ from

$$\texttt{r} := \texttt{trueNumber \% modulus}.$$

This is the all point: **if**

$$\texttt{observedNumber \% modulus} = \texttt{trueNumber \% modulus},$$

then adding the modulus to the value returned by Java would be fine. But in general, this is not the case!
The goal of the exercise is then to find a way to get r only looking at `observedNumber`.
By the generalized version of the distributive property of the % operation, we have

$$\begin{aligned}
&\texttt{Long.MAX_VALUE} + \texttt{valueOverflow} \ \ \texttt{\% modulus} \\
&= \big((\texttt{Long.MAX_VALUE} \ \ \texttt{\% modulus}) + (\texttt{valueOverflow} \ \ \texttt{\% modulus}) \big) \texttt{\% modulus} \\
&= (\texttt{remainderOfMax} + \texttt{remainderOverflow}) \ \ \texttt{\% modulus},
\end{aligned}$$

where

$$\texttt{remainderOfMax} = \texttt{Long.MAX\_VALUE} \ \ \texttt{\% modulus}$$

and

$$\texttt{remainderOverflow} = \texttt{valueOverflow} \ \ \texttt{\% modulus}.$$

Note that `remainderOfMax` + `remainderOverflow` is positive and less then `Long.MAX_VALUE` since by assumption $a \cdot$ `modulus` < `Long.MAX_VALUE` (one can definitely suppose $a > 2$) so it is not affected by overflows. This is thus the right correction to the overflow that we have to perform before applying %.
Now it only remains to get `valueOverflow` from the observed number (1), i.e.

$$\texttt{valueOverflow} = \texttt{observedNumber} - \texttt{Long.MIN\_VALUE} + 1.$$