# ICE 5 – Fibonacci Sequence

## Phase 2 – Initial Screen

Text

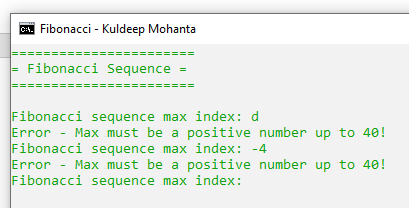
Description automatically generated

**Context:** When the JAR file initially starts up it displays “Fibonacci Sequence” as a welcome banner and it displays instructions for the user input (which in this case, it would be): “Fibonacci Sequence max index:”.

## Phase 3 – Input and Validation

Entering non-numeric input "d"


**Context:** In this input, the user had entered “d”. The application is not designed to input strings (or letters), so the application rejects the input with an error message and prompting the user with instructions on correct input.



**Context:** The user entered incorrect input, in this case it was “-4”. Which is not acceptable to the program, and the user is prompted to enter input with instructions for it being a “positive number up to 40”.

Text

Description automatically generated

**Context:** The user had entered a positive number but it was beyond the program’s pre-defined boundaries so it was not accepted. So, the user is prompted to enter input with instructions for it being a “positive number up to 40”.

## Part 4 – Output screens Graphical user interface, text, application Description automatically generated

Valid input #1 – Screenshot #5

“22”

Text

Description automatically generated with medium confidence  
Valid input #2 – Screenshot #6

“15”

**Question 1) What is a recursive method?**

In Java, a recursive method is a function that will use itself within itself to perform a task. It utilizes the function’s arguments to control the recursion as opposed to doing a similar task with a loop.

**Question 2) Just like loops, what could go wrong if recursion is used incorrectly?**

Among the top issues that can occur with failing to use recursion properly, some of the most notable ones are infinite recursion, loss of performance due to excessive program overhead, as well as naturally making an application harder to debug. As such it is important to use recursion correctly to maintain simplicity, efficiency, as well as understanding what problem recursion is being used to handle – since using a complicated concept on an easy solution could prove to be a nightmare in the long run.

**Question 3) How can recursive methods avoid the problem in Question 2?**

Probably the most simple, yet significant, solution to avoiding a bigger problem is to assess the original problem and the output’s beforehand. Being able to understand an applications inputs, outputs, and the simple processes that need to exist, can help to determine the needs of the program. In the end, providing clarity on if recursion is needed, and if so, then how to implement it efficiently and effectively.