Project 1: Abacus

DUE: Sunday, Feb. 9th at 11:59pm

Extra Credit Available for Early Submissions!

Setup

- Download the `project1.zip` and unzip it. This will create a folder `section-yourGMUUserName-p1`.
- Rename the folder replacing `section` with the `001`, `002`, `005`, etc. based on the lecture section you are in.
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address.
- After renaming, your folder should be named something like: `001-krusselc-p1`.
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`) .

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc.
- You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p1.zip` (no other type of archive) following the same rules for `section` and `username` as described above.
    - The submitted file should look something like this:

```
001-krusselc-p1.zip --> 001-krusselc-p1 --> JavaFile1.java
                                            JavaFile2.java
                                            JavaFile3.java
                                            ...
```

- Submit to blackboard. DOWNLOAD AND VERIFY WHAT YOU HAVE UPLOADED THE RIGHT THING. ***Submitting the wrong files will result in a 0 on the assignment!***

Basic Procedures

You must:

- Have code that compiles with the command: **`javac *.java`** in your user directory
- Have code that runs with the following command: **`java AbacusGUI full`**

You may:

- Add additional methods and variables, however these methods **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional public methods or variables.
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no **`ArrayList`**, **`LinkedList`**, **`HashSet`**, etc.).
- Use any arrays anywhere in your program (except when using/modifying the data field provided in the **`DynArr310`** class).
- Alter any method signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Alter provided classes that are complete (**`Abacus`**, **`AbacusGUI`**).
- Add any additional import statements (or use the "fully qualified name" to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

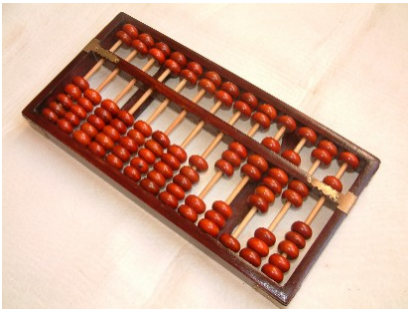
Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Overview

There are three major components to this project:

1. Implementing one the most fundamental data structures in computer science (the dynamic array list).
2. Using this data structure to implement a larger program.
3. Practicing many fundamental skills learned in prior programming courses (generics, class design, method overriding, comparisons, and many others).

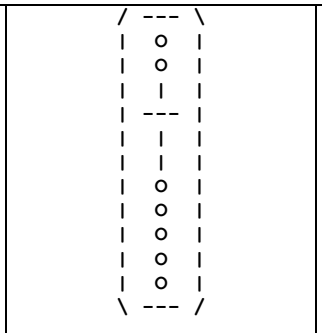
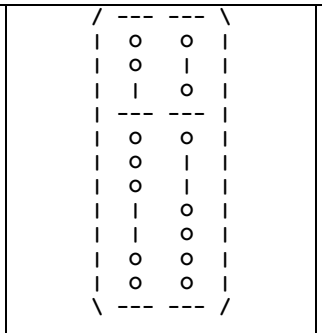
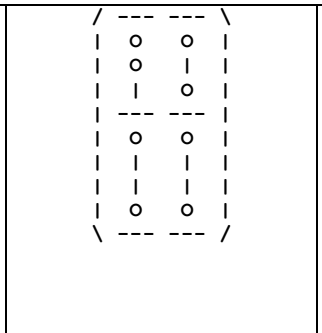
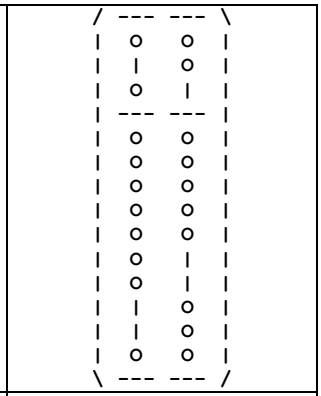


The end product will be a program showing the steps to add numbers with an abacus (picture shown on the right[1]). An abacus is an ancient tool for doing basic math, but it is also useful for training many "mental math" techniques. We will be simulating the steps for using a Chinese abacus (which has two beads on the top).

Never used and abacus before? This is a nice tutorial series that covers both types of abacus: https://www.youtube.com/playlist?list=PLVYm4hbKyOudk5Mjuw6pv_pKJmJIwDD1O
Remember that we're using the type of abacus with two beads on top and the exchange method.

A standard abacus with two beads on top and five on the bottom can do base 10 math and there are techniques for doing base 16. Out program, however, will be able to simulate abaci with different numbers of beads on the bottom for other even bases (such as base 2, 4, etc.).

Below are some examples to give you an idea of what you are doing. An annotated program run is shown at the end of this document. Note that the abacus for other bases always has two beads at the top (each representing the one unit of base/2) and base/2 beads on the bottom.

Abacus shown in ASCII art format:				
Num. top beads in use:	0	1s place: 1 10s place: 0	1s place: 1 4s place: 0	1s place: 0 16s place: 1
Num. bottom beads in use:	0	1s place: 1 10s place:3	1s place: 1 4s place: 1	1s place: 5 16s place: 7
Num. places:	1	2	2	2
Description:	A base-10 abacus representing the number 0 (no beads are pushed to the center). There is one column (representing the "1s place").	A base-10 abacus representing the number 36. There are two columns (representing the "1s place" and "10s place").	A base-4 abacus representing the number 13 (in base 4), or 7 (in base 10). There are two columns (representing the "1s place" and "4s place").	A base-16 abacus representing the number f5 (in base 16), or 245 (in base 10). There are two columns (representing the "1s place" and "16s place").

[1] Image source: https://en.wikipedia.org/wiki/File:Boulier1.JPG

Implementation/Classes

This project will be built using a number of classes representing the component pieces of the table we described in the previous section. Here we provide a description of these classes. Template files are provided for each class in the project package and these contain further comments and additional details.

- **`DynArr310` (`DynArr310.java`)**: The implementation of a dynamic array list for use by your other classes.
- **`MyAbacus` (`MyAbacus.java`)**: Your representation of an abacus that implements the **`Abacus`** interface.
- **`Abacus` (`Abacus.java`)**: An interface for classes which represent abaci. This class is provided to you and you should NOT change the file.
- **`AbacusGUI` (`AbacusGUI.java`)**: A GUI class for testing your abacus class. This class is provided to you and you should NOT change the file.

Requirements

An overview of the requirements are listed below, please see the grading rubric for more details.

- **Implementing the classes** - You will need to implement required classes and fill the provided template files.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods. Check provided classes for example JavaDoc comments.
- **Big-O** - Template files provided to you contains instructions on the REQUIRED Big-O runtime for many methods. Your implementation of those methods should NOT have a higher Big-O.

How To Handle a Multi-Week Project

While this project is given to you to work on over two weeks, you are unlikely to be able to complete this is one weekend. We recommend the following schedule:

- Step 1 (Prepare): Before the first weekend
  - Go over the project with a fine-toothed comb.
  - Make sure you understand everything you are going to be implementing.
  - Implement some of the **`DynArr310`** methods covered in class.
  - Learn to use an abacus if you're not familiar with them.
  - Think about how you want to represent your abacus (remember, you can't use Arrays!)
- Step 2 (**`DynArr310`** and **`MyAbacus`**): First weekend (1/31-2/2)
  - Implement the remaining methods from **`DynArr310`**.
  - Implement basic functions of your abacus (**`MyAbacus`**).
  - Try implementing **`add()`**.
- Step 3 (Finish Addition): Before the second weekend
  - Finish implementing addition with the abacus.

This schedule will leave you with the second weekend (2/7-2/9) to perform additional testing, get additional help, and otherwise handle the rest of life. ☺ Also, notice that if you get it done early in the week, you can get extra credit! Check our grading rubric PDF for details. Otherwise you'll have plenty of time to test and debug your implementation before the due date.

Testing

The main methods provided in the template files contain useful code to test your project as you work. You can use command like "**`java DynArr310`**" to run the testing defined in **`main()`**. You could also edit **`main()`** to perform additional testing. JUnit test cases will not be provided for this project, but feel free to create JUnit tests for yourself. A part of your grade *will* be based on automatic grading using test cases made from the specifications provided.

Once you think you have everything working, try different abacus bases with the AbacusGUI. You can use the command "**`java AbacusGUI full`**" to see the steps for addition or just use "**`java AbacusGUI`**" to see only the final results of the addition.

Example Run (Command Line)

```
>java MyAbacus
```

← Runs the main method in MyAbacus.

```
/ --- \
|  o  |
|  o  |
|  |  |
| --- |
|  |  |
|  |  |
|  o  |
|  o  |
|  o  |
|  o  |
|  o  |
\ --- /
```

← Starting abacus represents 0. An abacus always has at least one place (column). It should never have more columns than needed except when representing 0.

```
------------------------------
- Starting State
------------------------------
/ --- \
|  o  |
|  o  |
|  |  |
| --- |
|  |  |
|  |  |
|  o  |
|  o  |
|  o  |
|  o  |
|  o  |
\ --- /
```

← Printing out the results of adding 36.

The add() method returns all the "steps" needed to do the addition, including this starting state.

```
------------------------------
- Adding 36
------------------------------
/ --- --- \
|  o   o  |
|  o   o  |
|  |   |  |
| --- --- |
|  |   |  |
|  |   |  |
|  o   o  |
|  o   o  |
|  o   o  |
|  o   o  |
|  o   o  |
\ --- --- /
```

← 36 has more digits than we have columns, so the first "step" is to expand the abacus to the proper size.

```
/ --- --- \
|  o   o  |
|  o   o  |
|  |   |  |
| --- --- |
|  o   |  |
|  o   |  |
|  o   o  |
|  |   o  |
|  |   o  |
|  o   o  |
|  o   o  |
\ --- --- /
```

← Next we add the 30 from 36.

```
/ --- --- \
|  o   o  |
|  o   |  |
|  |   o  |
| --- --- |
|  o   |  |
|  o   |  |
|  o   o  |
|  |   o  |
|  |   o  |
|  o   o  |
|  o   o  |
\ --- --- /
```

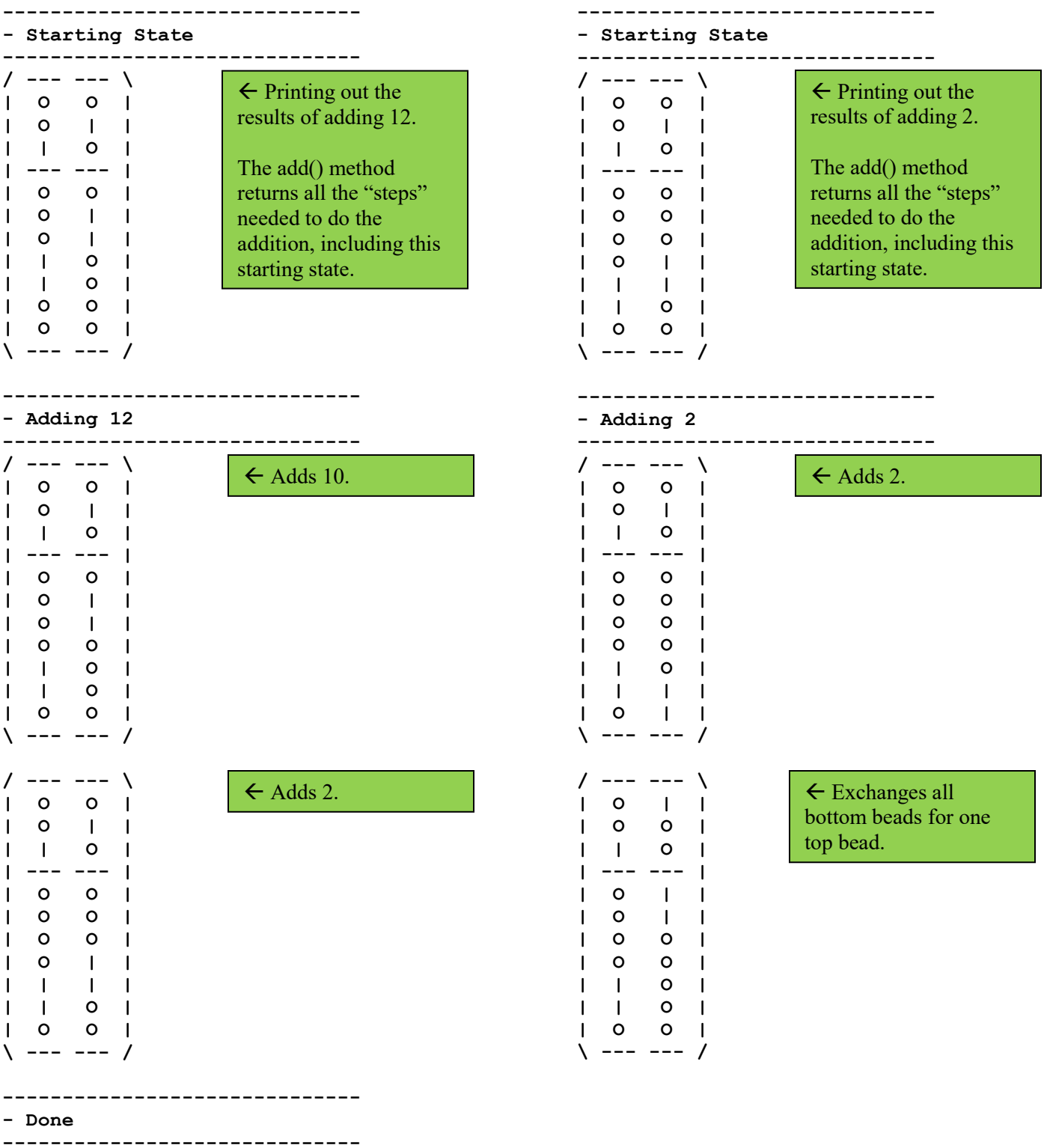
← Then add 5.

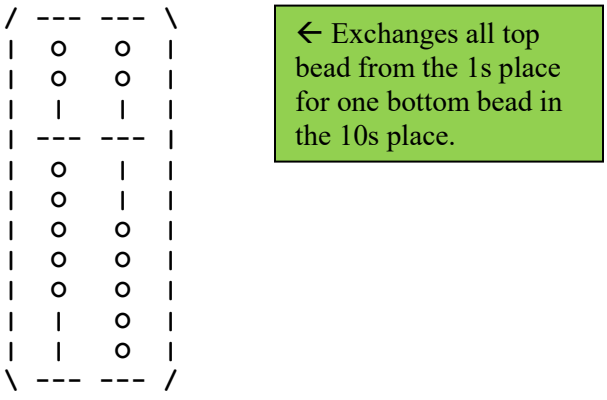
```
/ --- --- \
|  o   o  |
|  o   |  |
|  |   o  |
| --- --- |
|  o   |  |
|  o   |  |
|  o   o  |
|  |   o  |
|  |   o  |
|  o   o  |
|  o   o  |
\ --- --- /
```

← Then add 1.

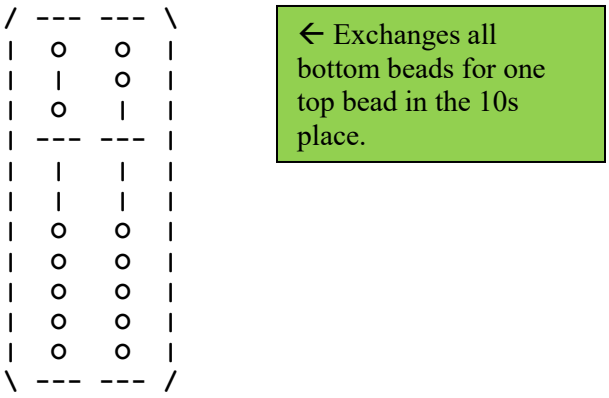
```
/ --- --- \
|  o   o  |
|  o   |  |
|  |   o  |
| --- --- |
|  o   o  |
|  o   |  |
|  o   |  |
|  |   o  |
|  |   o  |
|  o   o  |
|  o   o  |
\ --- --- /

------------------------------
- Done
------------------------------
```

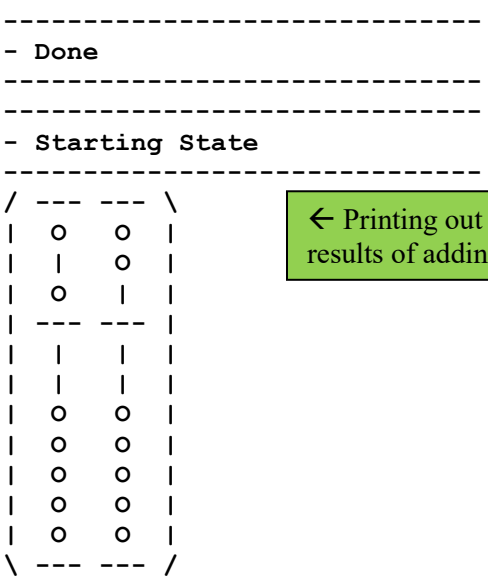

- Starting State
← Printing out the results of adding 12.
The add() method returns all the "steps" needed to do the addition, including this starting state.
- Starting State
← Printing out the results of adding 2.
The add() method returns all the "steps" needed to do the addition, including this starting state.
- Adding 12
← Adds 10.
← Adds 2.
- Adding 2
← Adds 2.
← Exchanges all bottom beads for one top bead.
- Done



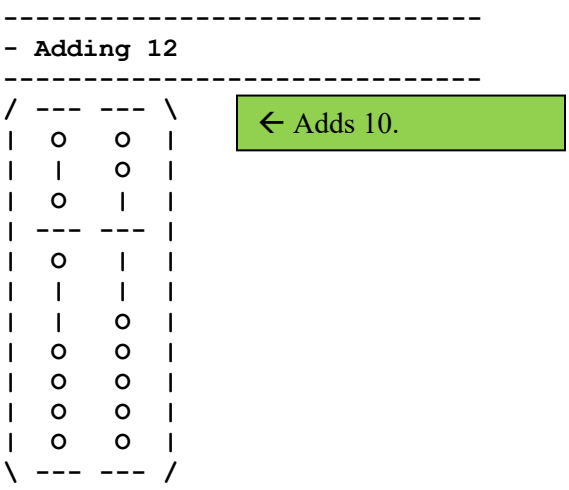
← Exchanges all top bead from the 1s place for one bottom bead in the 10s place.



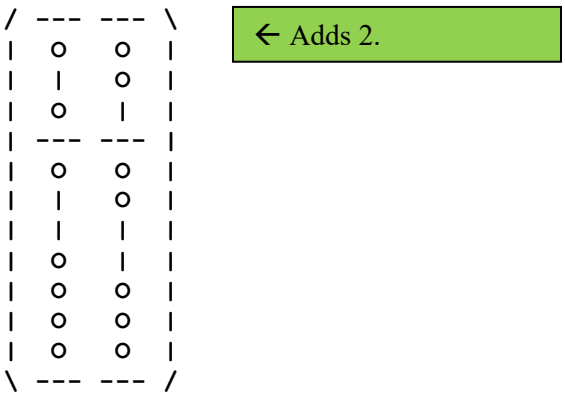
← Exchanges all bottom beads for one top bead in the 10s place.



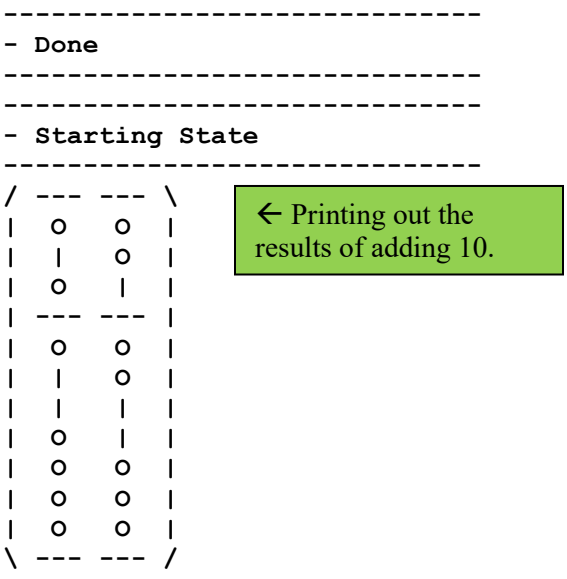
← Printing out the results of adding 12.



← Adds 10.

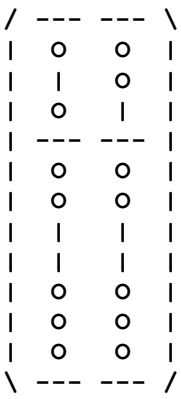


← Adds 2.



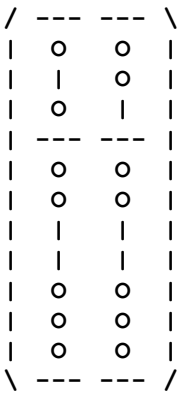
← Printing out the results of adding 10.

```
------------------------------
- Adding 10
------------------------------
```



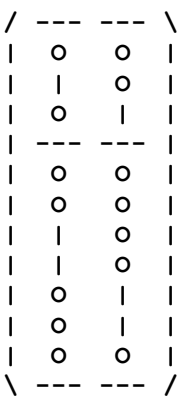
← Adds 10.

```
------------------------------
- Done
------------------------------
------------------------------
- Starting State
------------------------------
```



← Printing out the results of adding 2.

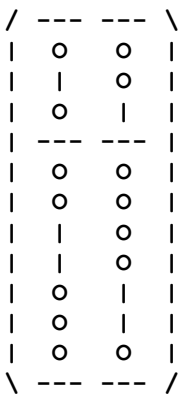
```
------------------------------
- Adding 2
------------------------------
```



← Adds 2.

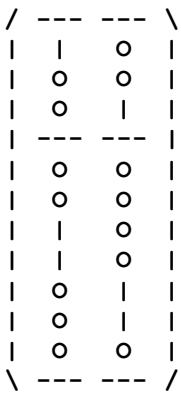
```
------------------------------
- Done
------------------------------
```

```
------------------------------
- Starting State
------------------------------
```

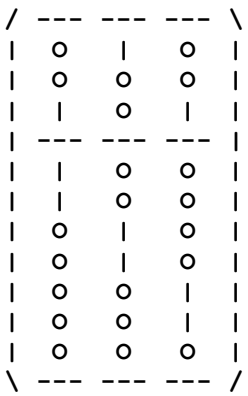


← Printing out the results of adding 68.

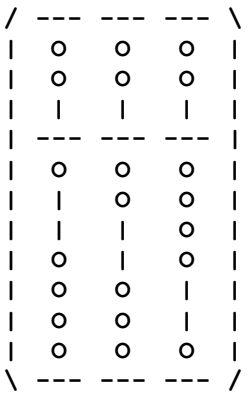
```
------------------------------
- Adding 68
------------------------------
```



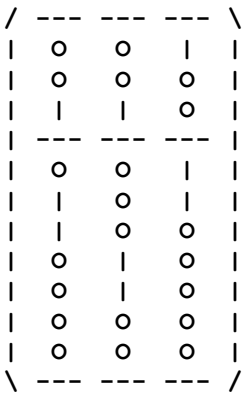
← Adds 50.



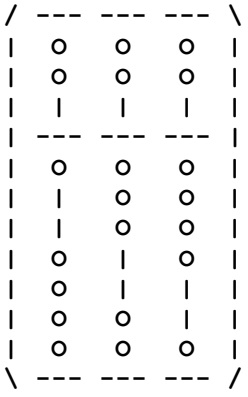
← Expand before exchange (need one more place to do the exchange).



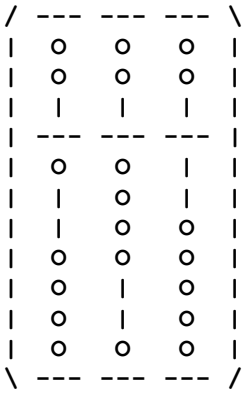
← Exchanges 2 50-beads for one 100-bead.



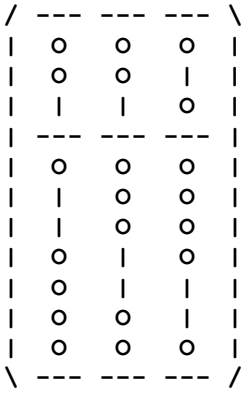
← Exchange.



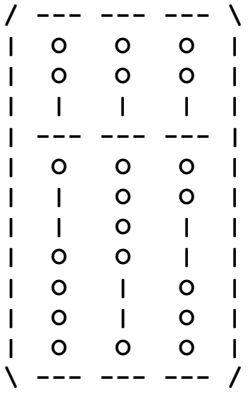
← Adds 10.



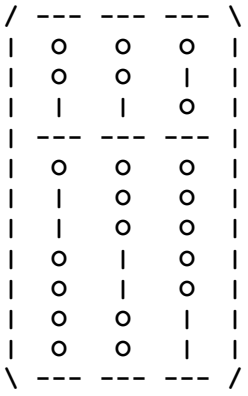
← Exchange.



← Adds 5.

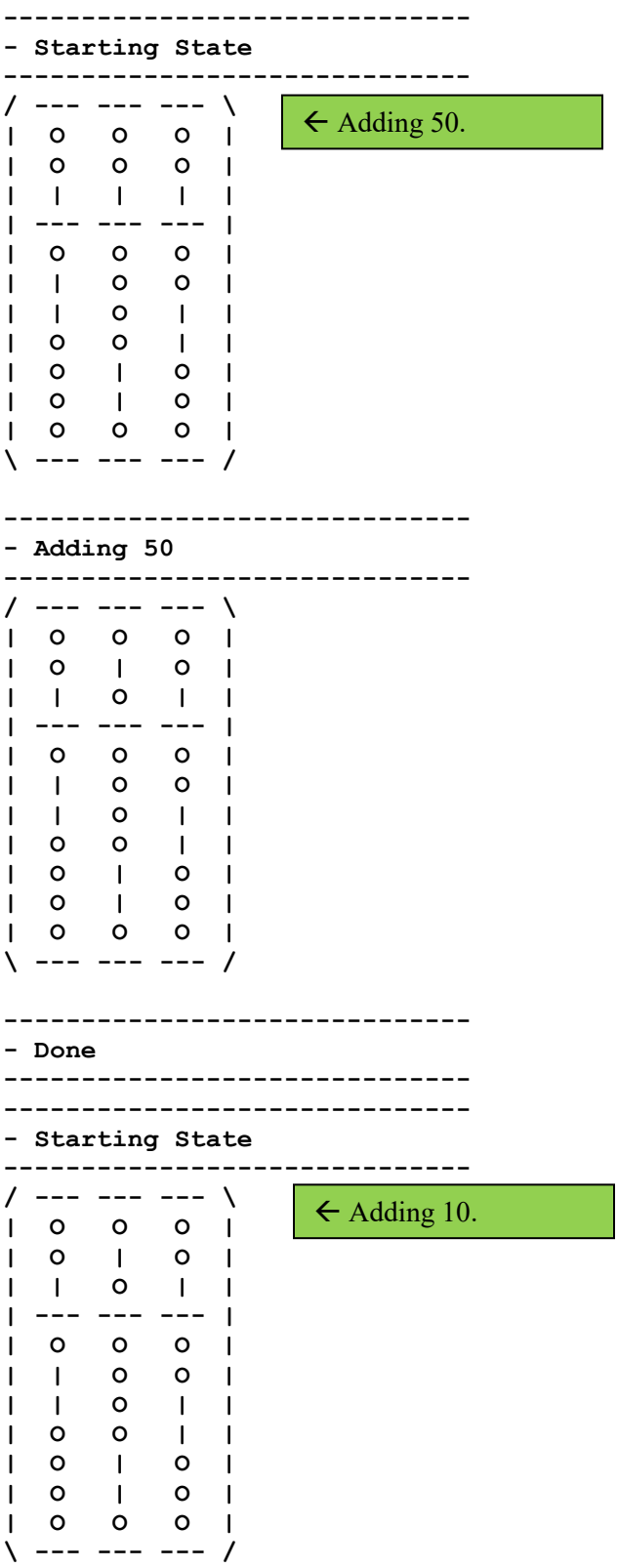


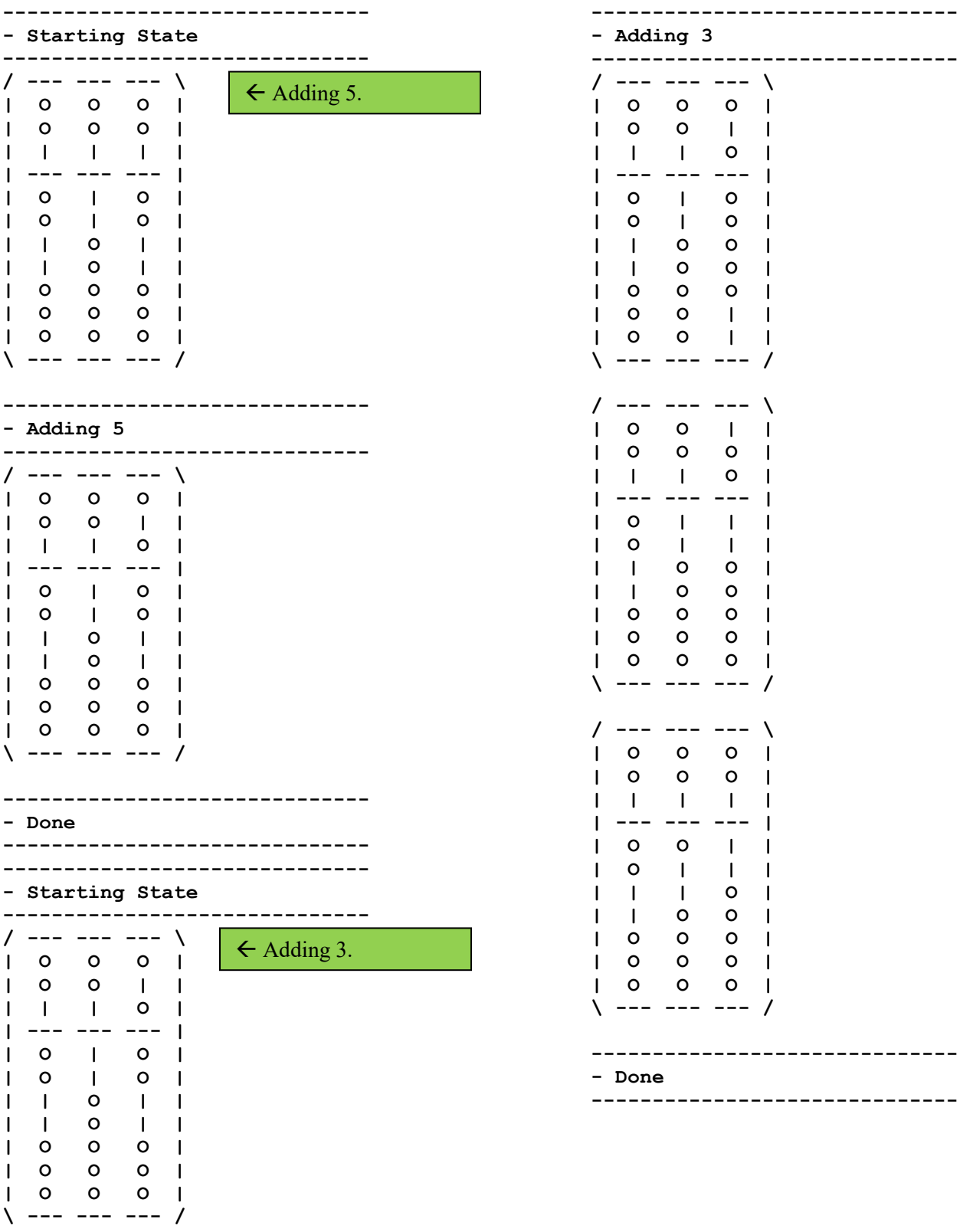
← Adds 2.

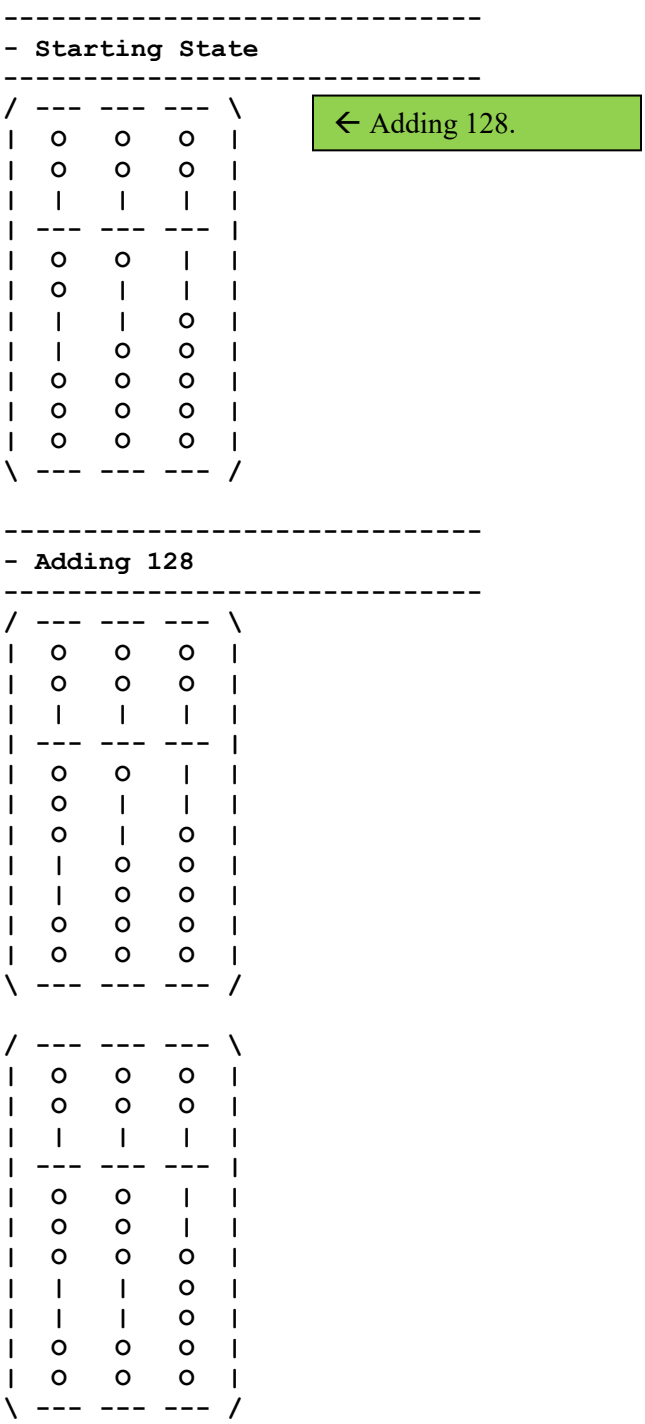


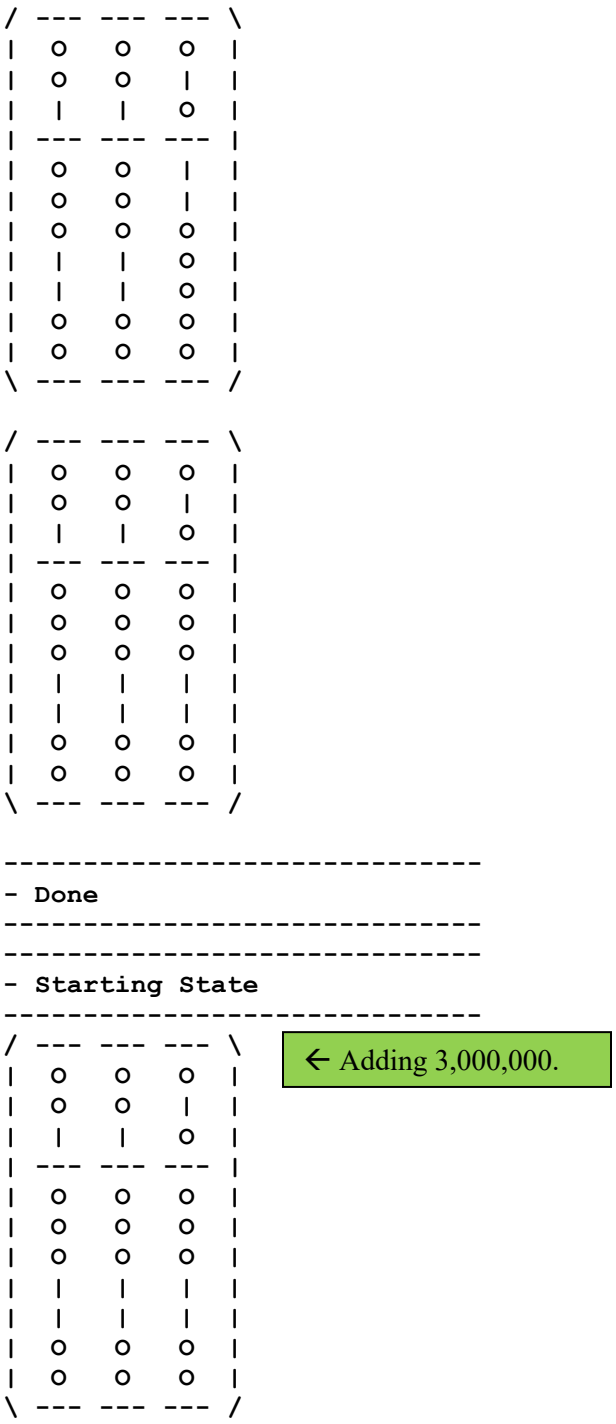
← Adds 1.

```
-----------------------------
- Done
-----------------------------
```


- Starting State
← Adding 50.
- Adding 50
- Done
- Starting State
← Adding 10.
- Adding 10
- Done


- Starting State
← Adding 5.
- Adding 5
- Done
- Starting State
← Adding 3.
- Adding 3
- Done


- Starting State
← Adding 128.
- Adding 128


- Done
- Starting State
← Adding 3,000,000.

```
------------------------------
- Adding 3000000
------------------------------
/ --- --- --- --- --- --- --- \
|  o   o   o   o   o   o   o  |
|  o   o   o   o   o   o   |  |
|  |   |   |   |   |   |   o  |
| --- --- --- --- --- --- --- |
|  |   |   |   |   o   o   o  |
|  |   |   |   |   o   o   o  |
|  o   o   o   o   o   o   o  |
|  o   o   o   o   |   |   |  |
|  o   o   o   o   |   |   |  |
|  o   o   o   o   o   o   o  |
|  o   o   o   o   o   o   o  |
\ --- --- --- --- --- --- --- /

/ --- --- --- --- --- --- --- \
|  o   o   o   o   o   o   o  |
|  o   o   o   o   o   o   |  |
|  |   |   |   |   |   |   o  |
| --- --- --- --- --- --- --- |
|  o   |   |   |   o   o   o  |
|  o   |   |   |   o   o   o  |
|  o   o   o   o   o   o   o  |
|  |   o   o   o   |   |   |  |
|  |   o   o   o   |   |   |  |
|  o   o   o   o   o   o   o  |
|  o   o   o   o   o   o   o  |
\ --- --- --- --- --- --- --- /

------------------------------
- Done
------------------------------
```