Matrix Calculator

Jonas Frankemölle, V. Corina, Luke Genna, Zach Boone

Purpose

User is able to:

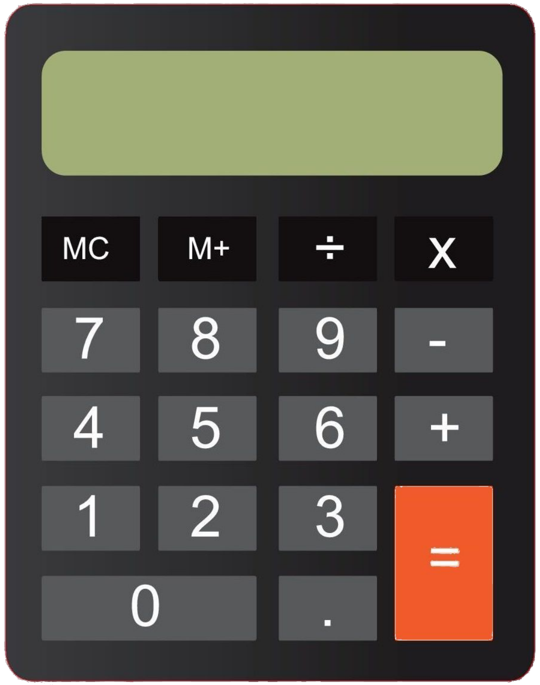
- Perform basic operations (addition, subtraction, etc.)
- Perform matrix operations (addition, subtraction, etc.)
    - Calculate determinant, dot product, cross product, transpose a Matrix
- GUI shows results
- Intended for people in need of a matrix based calculator

Technologies Used

- Java
- JUnit
- GitHub
- NetBeans
- Eclipse

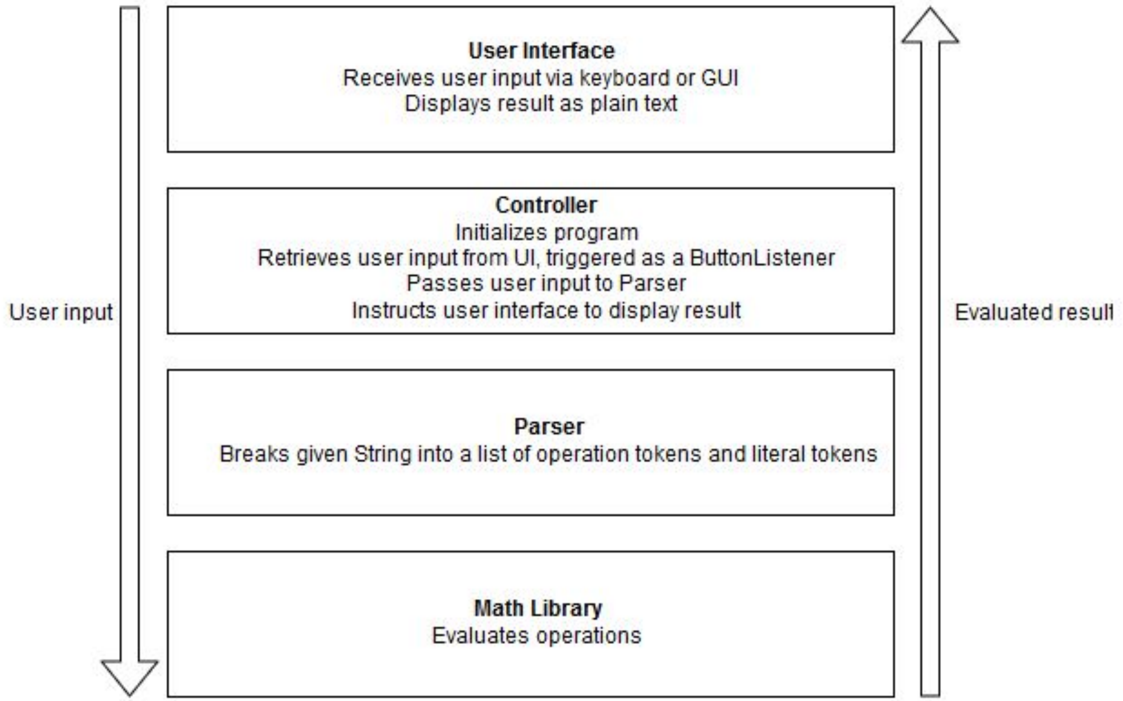
Demonstration

The demonstration station.



System Overview

- Four subsystems
  - UI
  - Controller
  - Parser
  - Math library
- Data flow


User Interface
Receives user input via keyboard or GUI
Displays result as plain text
Controller
Initializes program
Retrieves user input from UI, triggered as a ButtonListener
Passes user input to Parser
Instructs user interface to display result
Parser
Breaks given String into a list of operation tokens and literal tokens
Math Library
Evaluates operations
User input
Evaluated result

Parser

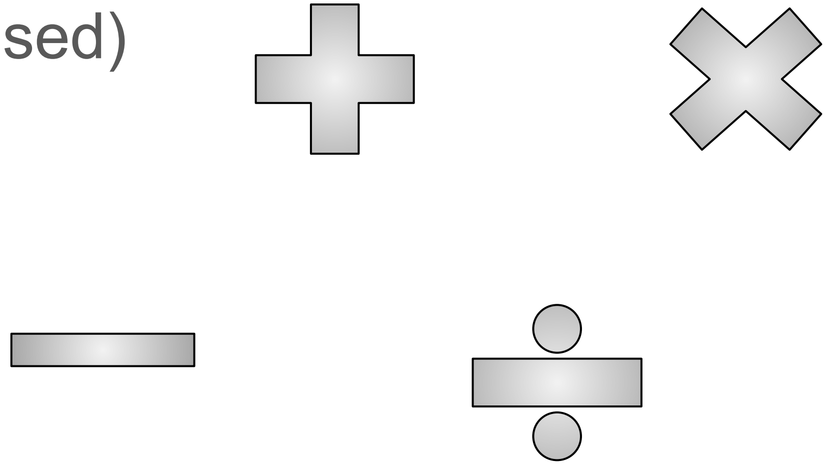
The parser uses a regular expression to split the input into an array of strings. It then converts those strings into literal and operator tokens.

As operations are created, they're added to a priority queue data structure where an operation's priority is its place in the order of operations (PEMDAS) modified by any parenthesis. All operations are then executed in order of priority. When an operation is performed, it inserts the result into the chain of Tokens.

By the end of the execution, all operations have been evaluated and there is a single literal remaining.

Math Functions

- Matrix math (add, subtract, multiply, transpose, power, …)
- Dot product (Vector Based)
- Cross product (Vector Based)
- PEMDAS calculations



View

- Java Based GUI developed with Swing
- Creates a Calculator on your screen
- Uses multiple listeners to interpret user input
- Input can come from either the user’s keyboard or from the buttons provided
- Output is displayed along the expression the user provided

Challenge: Integration

While we were able to communicate and make design choices as a group, development was independent to a fault. For much of development subsystems were written separately and we faced the challenge of integration with deadlines looming. We had some idea how everything would interact when writing it, but it still took some refactoring to make it all work.

If we were to do it again, we would be better off designing an agreed upon interface to create seams where subsystems met, as well as performing continuous integration.

Potential Improvements

- More graceful error handling
- More operations, like determinants and inversion
- More abstraction of the classes
- More thorough testing
- More diverse data types

Questions?