Calculator

Native Android App in Java

By Dominic Mayhew

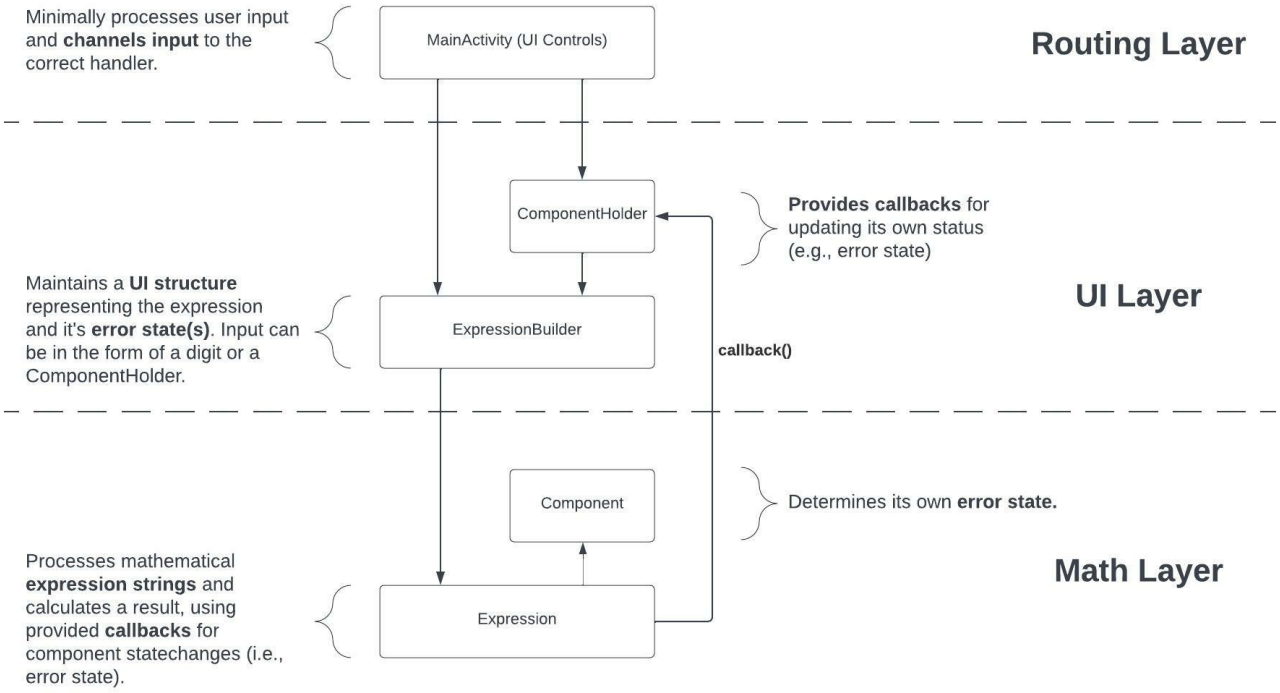
Purpose

To create an easy to use and understand calculator that provides live feedback on invalid input and allows the user to delete input and move the input cursor.

Requirements

1. **Easy** to use and read.
2. Makes the calculated **value of subexpressions** visible and intuitive.
3. Allow the **deletion** of input and allow **cursor** to move left and right.
4. **Highlight errors**. 3 categories of error:
   a) **Sequence errors** (e.g., two binary operators in a row). These errors can only be resolved by deleting the invalid input.
   b) **Arithmetic errors** (e.g., square root of a negative value).
   c) **Incomplete input errors** (e.g., a trailing binary operator). These errors are “resolvable” or temporary because they can be resolved without deleting anything
5. Includes a **headless version** of the math logic that can run on the JVM independent of the Android Framework.

Architecture



Routing Layer

- Minimally processes user input and **channels input to the correct handler**
- Passes input to the UI Layer through **digits** or **ComponentHolders**

UI Layer

- Maintains a **UI structure** representing the expression and its **error state(s)**
- UI structure is a nested list of layouts (i.e., containers for visual components).
- Passes input to the Math Layer in the form of **text** and state-change **callbacks**

Math Layer

- Processes mathematical **expression strings** and calculates a result
- Uses **callbacks** to communicate **changes in state** to the UI layer
- Expression is represented as a linked list of components (value components or operation components).

- Each component maintains its own error state and uses callbacks provided by UI Layer when a state change occurs.

What went well

1. It works! Mostly...
2. Using a linked list to represent the expression allowed **good encapsulation** of error state management, and made it really easy to implement the **movement of the cursor**.

Challenges

1. Lots of possible states to manage.
2. Parsing needs to continue even after an error has been encountered.
3. Incomplete errors.
4. Two analogous data structures
5. Duplication of logic
6. Evaluation algorithm

What I Learned

1. **Simplified interfaces** between layers.
2. My UI Layer and Math Logic Layer had **too much duplicated logic**.
    a) Next time, I would only have **one data structure**, maintained by the Math Logic layer.
    b) The UI Layer would provide **classes for individual components** and subexpressions with a defined interface for instantiation.
3. **Reconstruct** a tree data structure every time, but include a “dirty flag”.
4. **The UNIX principle**: “Do one thing and do it well.”

Future Development

1. Allow cursor movement by touching a part of the expression.
2. Non-linear display of expression components.
    a) E.g., division would result in displaying the operation as a vertical fraction.

Known Bugs

1. Entering an operator after evaluating an expression should initialize a new expression with the value of the previous expression at the start.
    a) E.g., “4+1=” followed by “*2” should create “5*2”
2. Some arithmetic errors do not resolve when they should.
3. Cannot handle inputting negative numbers.