# Design Summary of Scientific Calculator

1. Whole Structure
2. Design Details of Main Activity
3. Design Details of Draw Activity
4. Design Details of Calculate Core
5. Design Details of File Util

## Whole Structure

According to the requirement of the scientific calculator, we divide the whole App into 4 parts:

Main Activity, Draw Activity, Calculate Core, File Util.

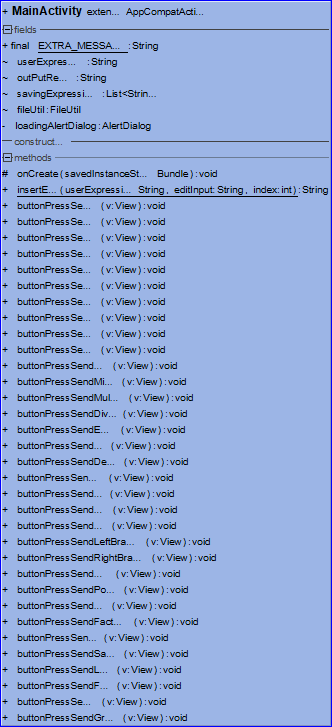
Main Activity is mainly about the GUI of the main page. It contains buttons for input and EditText area for showing inputs and results. Most of the actions are focus on message passing. As for simple expression, it can get the input button that pressed by user, update the information in the text area, generate the expression, send it to calculate core and get the result, and show the evaluated result in the text area. As for function expression, after getting input, it can ask CalculateCore to check the validation of the function expression and send it to Draw Activity for drawing the image. As for save/load, it can send userExpression to FileUtil to save in file, and pop out a diagram contains the list of loaded expressions provided by FileUtil.

Draw Activity is designed for drawing function imagine. It contains a canvas to show the image. Actions in it are utilities for dealing with given function expression and generate the points which can plot on canvas to show the function line.

Calculate Core handles all of the tasks about evaluating an expression and generating, managing and evaluating a function. It contains Token, Tokenizer and Expression classes, which are designed for parsing expressions and functions from string.

File Util is focus on saving and loading expressions inputted by user. This part gets the expression from Main Activity and saves it to file. It also has the ability to load up to five expressions from history file, transform it to a list of String and provide the list to Main Activity.

## Design Details of Main Activity

 UML of MainActivity.class

Since the MainActivity is directly related to the main interface, we can see most of the methods are handlers for buttons. When the MainActivity noticed some button is pressed, it will change the userExpression field and the text in the EditText. Below few special methods will be explained:

**Fields:**

|  |  |  |
| --- | --- | --- |
| Type | Field | Description |
| String | userExpression | The expression synchronized with EditText, it is the message passed to DrawActivity and CalculateCore |
| String | outPutResult | The evaluated result shown to user |
| List<String> | savingExpressions | The list storing expressions which can be stored to files |
| FileUtil | fileUtil | The file utility used for saving and loading expressions |
| AlertDialog | loadingAlertDialog | The dialog shown when user click the load button. It provides a list for user to choose which expression to use. |

**Methods:**

|  |  |  |
| --- | --- | --- |
| Type | Method | Description |
| String | insertEdit(String userExpression, String editInput, int index) | put input from EditText through moving cursors to the expression |
| Void | buttonPressSend0(View v) | Handler for 0 button. Change the userExpression and EditText.  (Most of ButtonPressSend methods are similar, so they will not be shown here.) |
| Void | buttonPressSendEqual(View v) | Handler for = button. Send the userExpression to CalculateCore and get the result. Update the outPutResult. |
| Void | buttonPressSendDelete(View v) | Delete 1 character in EditText and userExpression. |
| Void | buttonPressSendAC(View v) | Delete all contents in userExpression, outPutResult and EditText. |
| Void | buttonPressSendSaved(View v) | Save savingExpressions to file. |
| Void | buttonPressSendLoad(View v) | Load expressions from file. Generating a AlertDialog containing a list of saved expressions for user to choose. Set userExpression and EditText as the expression choosed by user. |
| Void | buttonPressSendGraph(View v) | Ask Calculator to check whether the userExpression is a valid function. If it is, send it to grid and start the DrawActivity. |

## 图片包含 屏幕截图 描述已自动生成Design Details of Draw Activity

UML of grid

The grid class and related layout xml implemented the DrawActivity. All the fields and methods are designed for making the data array suitable for drawing on different screen and drawing them.

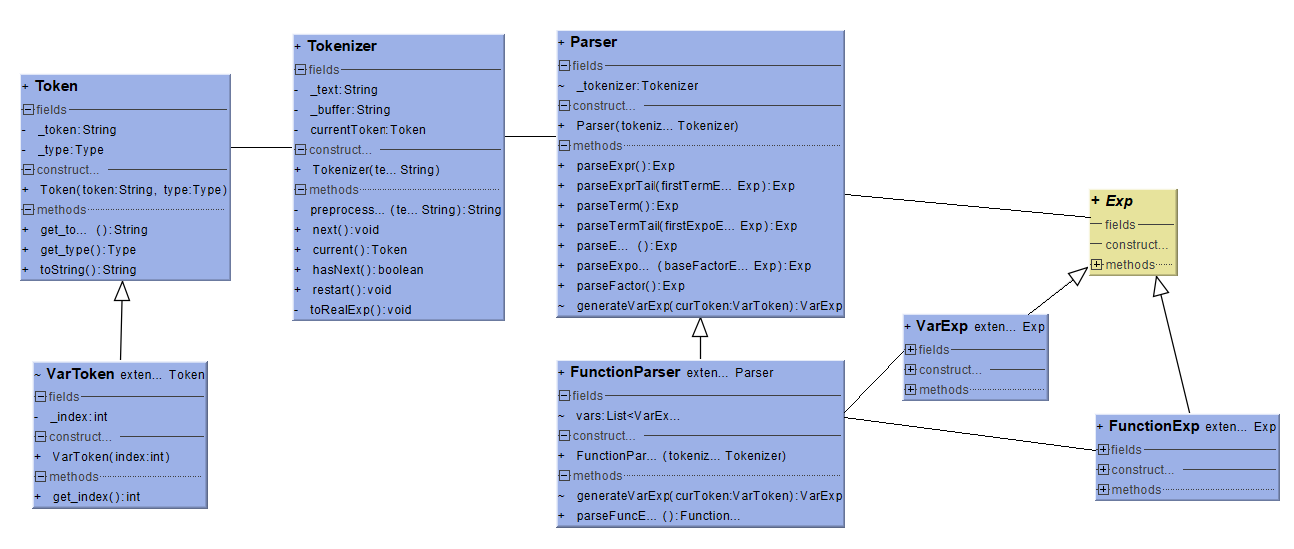
**Fields:**

|  |  |  |
| --- | --- | --- |
| Type | Field | Description |
| float[] | pointY | Storing the y of points gotten from CalculateCore |
| String | functionExp | Storing the expression gotten from MainActivity |
| float | maxValue | Storing the max value of pointY |
| float | widthParameter | Data of the drawing canvas |
| float | heightParameter | Data of the drawing canvas |
| float | diagramWidth | Data of the drawing canvas |
| float | diagramHeight | Data of the drawing canvas |
| float | diagramLeft | Data of the drawing canvas |
| float | diagramTop | Data of the drawing canvas |

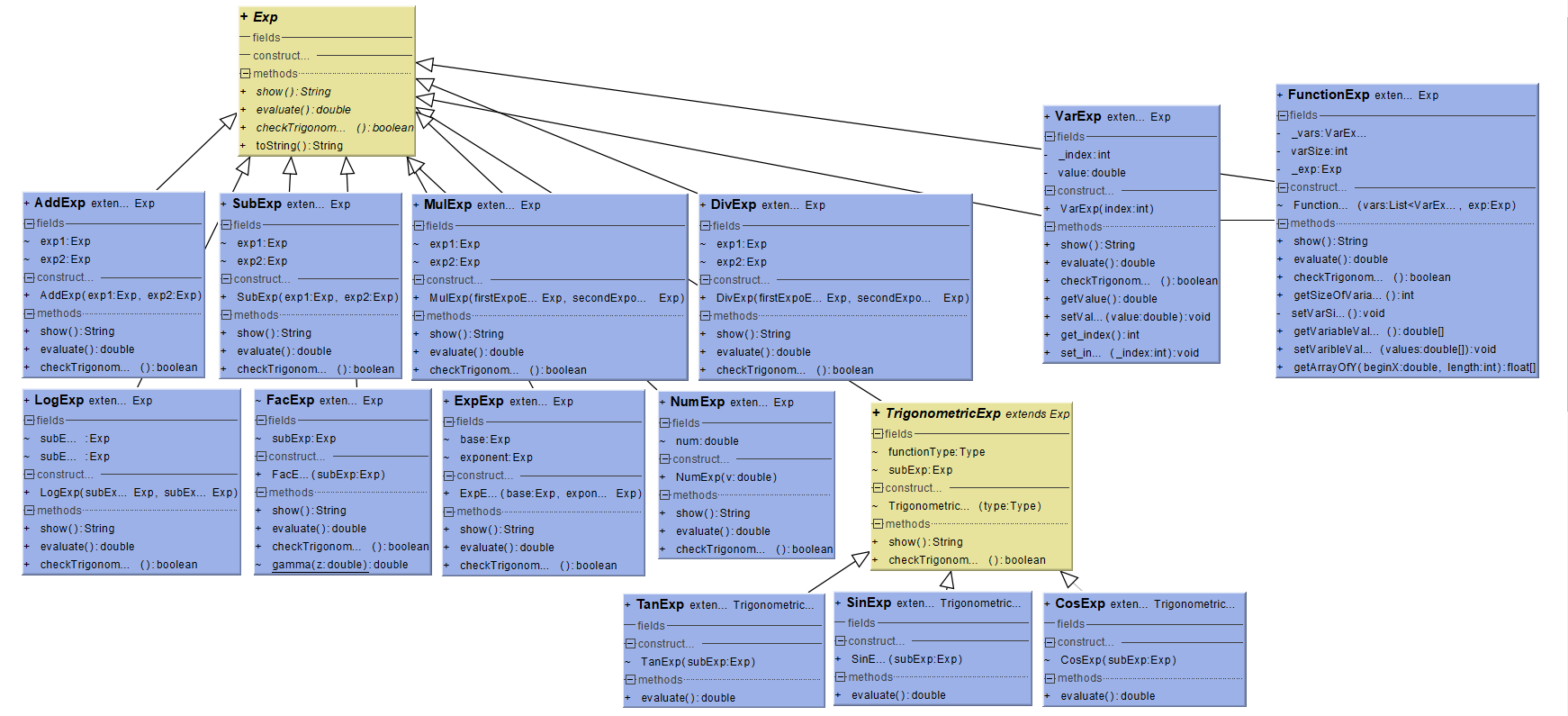
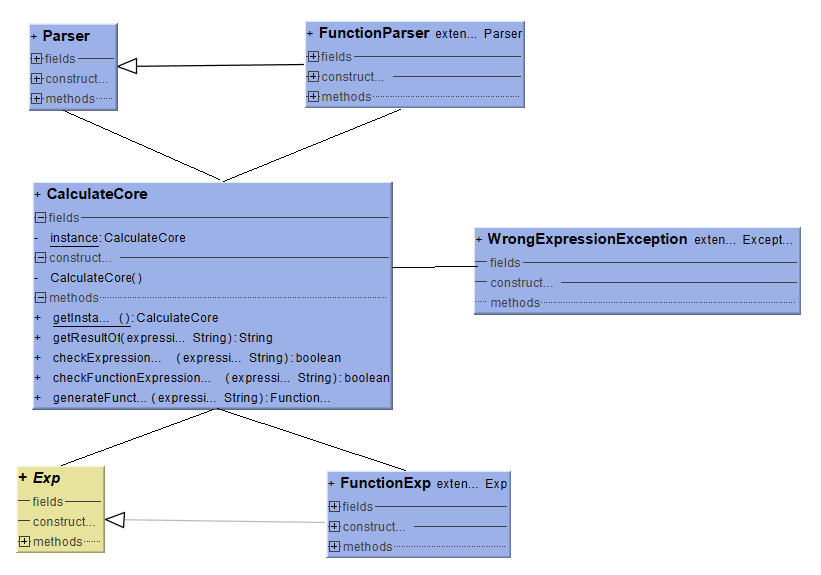
**Methods:**

|  |  |  |
| --- | --- | --- |
| Type | Method | Description |
| Void | onDraw(Canvas canvas) | Handler of drawing. |
| Void | Initialize(Canvas canvas) | Send function and the range of X to Calculate Core, get a array of Y values. |
| Void | Print(Canvas canvas) | Set a paint object. |
| Void | renderDiagram(Canvas canvas, Paint paint) | Draw the curve. |
| float | regulate(float x) | Make the value of x suitable for drawing |
| float | regulate(float y) | Make the value of y suitable for drawing |

## Design Details of Calculate Core

The Calculate Core part is combined with Expression System, Parser System and an API, CalculateCore.

UML of Parser System

UML of Expression System

UML of Calculate Core

Since it is a complex system, some details are ignored in this summary.

Parser, Tokenizer and Token are classic design for passing an Expression. Tokens represent different parts of an expression. The tokenizer parses a string into a series of Tokens. With the Tokenizer, the Parser can generate a tree-like data structure of Expressions.If the expression user typed is wrong, it will throw a WrongExpressionException. Expression class provides methods to evaluate, show the result of the expression. To parse function, which is kind of different from common expression, VarToken, FunctionParser, VarExp, FunctionExp are designed as child classes of Token, Parser, Exp. VarToken has one extra field to state the index of var. FunctionParser has extra fields and methods to manage the variables in the function. VarExp makes VarTokens a part of an Expression so that the FunctionExp can evaluate with different values of variables. The FunctionParser generate a FunctionExp after parsing instead of a simple Exp. The parsing gramma is shown below:

Expr -> Term ExprTail

ExprTail -> + Term ExprTail

| - Term ExprTail

| null

Term -> Expo TermTail

TermTail -> \* Expo TermTail

| / Expo TermTail

| null

Expo -> Factor ExpoTail

ExpoTail -> ^ Factor ExpoTail

| null

Factor -> (Expr)

| Functions1(Expr)

| Functions2(Expr)(Expr)

| NUM

| VAR

Functions1: SIN, COS, TAN, FAC

Functions2: LOG

Expression System is combined by Exp class and its subclasses. Different Exp has different amount of subclasses. For example, AddExp has 2 subExp and SinExp has1 subExp. All of them have three same methods:

|  |  |  |
| --- | --- | --- |
| Type | Method | Description |
| String | show() | Output the Exp as a string with brackets showing priority. |
| Double | evaluate() | Evaluate the Exp according to mathematics. |
| boolean | checkTrigonometric() | Check whether the Exp has a part like “sin(x\_1)”. It is for better look when drawing the curve of Trigonometric. |

CalcuateCore class is a programing interface for other part of the App. It designed by the singleton pattern. It handles the exceptions inside and give out an error string if user put in a wrong expression. Below are details of methods:

|  |  |  |
| --- | --- | --- |
| Type | Method | Description |
| CalculateCore | getInstance() | Return an instance of CalculateCore. |
| String | getResultOf(String) | Calculate a string expression and return the value. If the expression wrong, it will return “ERROR, WRONG EXPRESSION” |
| boolean | checkExpressionValid(String) | Check whether a string expression is correct. |
| boolean | checkFunctionExpressionValid(String) | Check whether a string function expression is correct. |
| FunctionExp | generateFunction(String) | Parse a string function expression into a FunctionExp object |

## Design Details of File Util

UML of FileUtil

The FileUtil is focus on saving and loading String type Expressions.

**Fields:**

|  |  |  |
| --- | --- | --- |
| Type | Field | Description |
| String | FILE\_NAME | The name of saving/loading file. |
| Context | context | The context of the Activity using FileUtil. |

**Methods:**

|  |  |  |
| --- | --- | --- |
| Type | Method | Description |
| void | Save(List<String>) | Save the String List to the file. Since we limit the size of list as up to 5, when the size is six, we just delete the oldest one. |
| List<String> | loadFile() | Load the Strings from file. |