|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Method** | **Parameters** | **Variables** | **Class** | **Description** |
| BODMAS | The mathematical expression that the user enters that will be calculated. This is called the infix. | * infix * postfix | ShuntingYard | This method is the basis of the entire app. This will take a string that the user provides, called the “infix”.  I will use the Shunting Yard algorithm to apply BODMAS. This algorithm is stack based and works by taking the infix (which is the user’s mathematical expression) and converts it into the postfix. The postfix is created by manipulating the infix with stacks. The postfix is also known as Reverse Polish Notation or RPN.  The postfix can then be evaluated to compute an answer to the user’s original expression. This will occur in a different method.    For example, if the infix is “3 + 4 × 2 ÷ ( 1 − 5 ) ^ 2 ^ 3” the postfix will become “3 4 2 × 1 5 − 2 3 ^ ^ ÷ +”. |
| EvaluateRPN | The postfix that has been created by the BODMAS function. | * postfix * result | ShuntingYard | This method will evaluate the postfix and return the answer to the user’s original expression.  This is done by looping through the string of the expression and when the specific element is an operator, pop two numbers off the stack and manipulate them according to the type of operator that is given.  For example, if a “+” operator is the current element in the loop, two numbers will be popped off the stack and added together. This works similarily for other basic operations such as powers, division, subtraction, etc.  Because the string is in Reverse Polish Notation, the order of the numbers and operators means that when the numbers are popped off the stack, they will be calculated according to BODMAS. |
| Roots | A float as the base and an integer as the root. | * base * root * result | MainActivity | The user can enter their root number and their base number. The base number will be rooted by the root using the Java Math class.  The result will then be stored in the result variable.  For example, the user entering “2” and “4” will perform a square root of 4. |
| Trigonometry | Which function is to be used (sin/cos/tan).  The number that it should be performed on. | * trigFunction * input * result | MainActivity | The method will evaluate the trigFunction variable to decide the trigonometry function that will be used.  Depending on the trigonometry function, the user’s input will be calculated using Java’s Math class. The answer from this will be stored in the result variable. |
| Inverse Trigonometry  (sin-1, cos-1, tan-1) | Which trigonometric function is to be used (sin-1/ cos-1/tan-1).  The number that it should be performed on. | * trigFunction * input * result | MainActivity | The method will decide which inverse trigonometric function to use on the user’s input depending on the parameters given to the method.  When the calculation is done, the result will be stored in the result variable.  This method will require validation so the user’s input are within the appropriate bounds. For example, doing the calculation sin-1(x) where -1 ≤ x ≤ 1 will result in an error. Therefore, the method will require validation to prevent the user performing these actions and crashing the app. |
| convertNotation | Which notation form the user wants to convert their answer to.  The notation that the user’s number is currently in.  The number that the conversion will be performed on. | * currentNotation * requiredNotation * input * result | MainActivity | This method will take the user’s input and convert it to all the other notation forms. This will be displayed to the user.  For example, if the user has a decimal number, it will be shown in fractions, standard form, and surds.  The user then has the option to select a specific type of notation for their input to displayed in. |
| convertUnits | A string containing the current unit and a double containing the value of that unit. | * currentUnit * input * resultNum * resultUnit | MainActivity | This method will be called on the user’s click of a button. It will bring up a menu where the user can select the conversion they want. For example, they can select “cm -> m”.  The method will either take the number currently stored in the calculator’s answer variable or the user can enter their own.  In this example, where the user has just entered a calculation with the result of “120.0”, the currentUnit would be “cm”. The input is “120.0”, the resultUnit is “m”.  The calculator will then perform the conversion of centimetres to metres and store the result in the variable “resultNum”.  In this example, the resultNum would be “1.2”.  The method will have the functionality to convert:   * distance * time * area * mass * volume * speed |
| Display Error | The error code | * errorCode * errorDetail | MainActivity | This method will be called for different types of errors.  For example, it will be called when the calculator encounters a failed verification check. When this occurs, the error code will be a parameter in the method call.  This error code will be linked to a description of the error and how to fix it. This description will be a string stored in the variable “errorDetail” and will be displayed to the user.  Another example of when this method might be called is when a method catches an exception. Each type of exception will have its own unique error code with a corresponding description of the error that will be displayed to the user. |
| showEquations |  | * selectedEquation | Equations | This method will display a list of the stored mathematical equations in the calculator.  When the user selects one, this method will call “useEquation(x)” where x is the user’s equation. |
| useEquation | The equation that the user has selected | * Variables that are used in the equation, such as frequency, distance, speed, time, etc.   These will be provided by the user.   * result | Equations | Upon being called, this method will query the user for to provide the known variables and select the unknown variable.  For example, with the equation to calculate speed/time/distance, the user would provide two variables – “30 metres” and “5 seconds”. Then the user would select solve for speed and the calculator would output “6 metres per second”.  All equations require the values to be entered in SI units. This is because the equations in GCSE and A Level papers are to be used with SI units. Therefore, this will help the user become used to doing calculations and become better prepared for their exams. |
| changeLayout | The new layout that the user has selected. | * currentLayout * tablet * phone * portrait * landscape | MainActivity | This method will change the xml layout of the GUI. This will accommodate different screen sizes, so all users can use the app easily without elements being cut off or inaccessible.  This method will also enable the users to switch between portrait and landscape layouts. The different orientations will require separate xml files because each one will need to be designed for the all the features to be intuitively accessible. |
| showTip | A variable that identifies which function the tip should be shown for. | * tipDetail | MainActivity | When a method is accessed for the first time, it will call this method.  This method will display a pop up to the user that explains the method that the user is accessing does and anything else that will help the user to use the app. The content of this will be stored in the variable “tipDetail”.  For example, when the user first accesses the convertNotation() method, a pop up will be shown explaining that the method shows different notations for the user’s results. It will also outline how the user can select a notation to be used in the calculator. |
| moveCursor | The directional input (this will be decided depending if the user clicks the left or right arrow). | direction | MainActivity | Moves the cursor in the GUI different direction depending on the user’s input. |