**Expression:**

* string input: Attribute for holding the entire input into the calculator program
* int finalResult: Attribute for holding the final result of the entire expression input into the calculator program
* number\*[] numbers: An array of pointers to number objects, where each pointer points to the number objects representing each of the different numbers in the input expression
* char[] operators: An array of operator characters, each index being represented by a different operator in the input expression
* int priorityIndex: An integer value that holds the index of the next operator (in the char[] operators array) to be performed based on the appropriate order of operations
* int getPriorityIndex(): Method that will parse the char[] operators array and find the index of the next operation to be performed, returning that index from the function
* int getExpressionResult(): Method for returning the result of a particular operation within the expression (will create one of the Addition, Subtraction, Multiplication, Division objects for every operation depending on what operation needs to be performed, unless there is only one number object remaining in the expression, in which case the final number value will be stored in the ‘finalResult’ integer attribute)
* void adjustArrays(): Method that will readjust the ‘numbers’ and ‘operators’ arrays after one operation is performed to ensure proper functionality of the ‘getExpressionResult()’ method
* void parseInput\_Nums(): Method that will parse the ‘input’ string, retrieve the numbers, and create a new number object for every number in the expression, storing a pointer to each of those objects in the ‘numbers’ array
* void parseInput\_Ops(): Method that will parse the ‘input’ string, retrieve the operators, and add each of those operators (in their respective input order) into the ‘operators’ array
* void setInput(): Method for setting the ‘input’ string value from the program into the ‘Expression’ class object
* int getFinalResult(): Method for retrieving the ‘finalResult’ attribute from the ‘Expression’ class object to be displayed to the user

**Number:**

* int value: Attribute that holds the integer value of the number object
* int getValue(): Method for returning the integer values held by the ‘value’ attributes of the different number objects (for accessing the different values to be operated upon)
* void setValue(): Method for setting the integer values held by the different number objects

**Addition:**

* int sum: stores the result of the numbers being added together
* void Add(): adds the two numbers passed into the function
* int getSum(): returns the value in sum to the user

**Subtraction:**

* int difference: stores the result of the numbers being subtracted
* void Subtract(): subtracts the two numbers passed into the function
* int getDifference(): returns the value stored in difference to the user

**Multiplication:**

* int product: stores the result of the numbers being multiplied together
* void Multiply(): multiplies the two numbers passed into the function
* int getProduct(): returns the value stored in product to the user

**Division:**

* int quotient: stores the result of the numbers being divided
* void Divide(): divides the two numbers passed into the function
* int getQuotient(): returns the value stored in quotient to the user

**Operation:**

* int left\_number: Attribute for holding the ‘left’ integer value of the particular operation to be performed (inherited by Addition, Subtraction, Multiplication, and Division classes)
* int right\_number: Attribute for holding the ‘right’ integer value of the particular operation to be performed (inherited by Addition, Subtraction, Multiplication, and Division classes)
* void setLeft(): Inherited method for setting the ‘left\_number’ attribute so that the proper operation can be performed on two integer values
* void setRight(): Inherited method for setting the ‘right\_number’ attribute so that the proper operation can be performed on two integer values
* int getLeft(): Inherited method for retrieving the ‘left\_number’ attribute’s value so that the proper operation can be performed in the Add(), Subtract(), Multiply(), or Divide() methods
* int getRight(): Inherited method for retrieving the ‘right\_number’ attribute’s value so that the proper operation can be performed in the Add(), Subtract(), Multiply(), or Divide() methods

**Summary of Implementation:**

The ‘Expression’ class will be the class that holds the entire expression entered into the calculator program. Every expression needs to have at least one number in order for the calculator to have an appropriate output, which is why the ‘Number’ class is a composition of the ‘Expression’ class (and also why the ‘Number’ class has a minimum multiplicity of 1 for every expression). There can be as many numbers as possible entered into the calculator program, so there is no maximum multiplicity for the ‘Number’ class.

The ‘Addition’, ‘Subtraction’, ‘Multiplication’, and ‘Division’ classes are all aggregations of the ‘Expression’ class, as each of these classes can (and most likely will) be present in an entered expression, but don’t necessarily need to be a part of the expression, hence why they are aggregations of the ‘Expression’ class. As none of these 4 general operation classes are required, they will all have a minimum multiplicity of 0 for every expression. However, as there can be as many of one operation as possible entered into the expression, all of these general operation classes will have no maximum multiplicity for an expression. Each of the 4 classes for each of the 4 operations the calculator can perform will inherit from a general ‘Operation’ class. These operation classes will inherit a left and right number from the abstract ‘Operation’ class, as well as methods to set and get those integer values.