1) Compiling the code

The required code is in Calculator.java.

To compile, simply go to command line and type

```
<Folder containing code> javac  Calculator.java
```

(This needs java tools to be added to the path variable.)

2) Executing the code

The program mandatorily takes in 4 arguments, each separated by a space:-

i) Argument 1 is the input String.
ii) Argument 2 is the INFO option "-- info"
iii) Argument 3 is the ERROR option "-- error"
iv) Argument 4 is the DEBUG option "-- debug"

Example:-

```
<Code Location> java Calculator "add(2, 5)" "-- info" "-- error" "-- debug"
```

If some verbosity options are not needed, they can be replaced with ""

Example:-

```
<Code Location> java Calculator "mult(2, 5)" "" "-- error" ""
```

Note that even if no verbosity is needed, we still need to give arguments

Example:-

```
<Code Location> java Calculator "mult(2, 5)" "" "" ""
```

3) Algorithm:-

i) We start parsing the input command to check if it has ','

- This will determine if the command is to be processed.
- If it doesn't have ',' we directly try convert it to Integer and return

ii) If the input command has ',' it means we need to process it.

- Now, we look for a good evaluation point, which is a ',' that has number before and after it.
- On finding the good evaluation point we look for the operation to be performed on these numbers.
- Next after evaluating this expression, we replace expression by evaluated value in the command and call the function again with the new command.

iii) If we have ',' but no good evaluation point, we look for let statements

- Traverse through the command to find good let statement (which has a variable and a number value next to it)
- Now replace all variable in let statements expression by its value.
- Form new command by replacing this let statement in the command by the above expression.
- Call the function again with this new command.

1) Sample input:
```
java Calculator "let(a, 5, let(b, mult(a, 10), add(b, a)))" "-- info" "-- error" "-- debug"
```

Sample output
```
Info: String to be parsed after eliminating spaces is
let(a,5,let(b,mult(a,10),add(b,a)))
Debug: Traversing for possible let statements in let(a,5,let(b,mult(a,10),add(b,a)))
Debug: Evaluating a let statement
Debug: Replacing variable a with value 5
Debug: New command after processing let is let(b,mult(5,10),add(b,5))
Debug: Reached a good evaluation point
Debug: Calling mult with parameter 5 and 10
Debug: New command after evaluation is let(b,50,add(b,5))
Debug: Traversing for possible let statements in let(b,50,add(b,5))
Debug: Evaluating a let statement
Debug: Replacing variable b with value 50
Debug: New command after processing let is add(50,5)
Debug: Reached a good evaluation point
Debug: Calling add with parameter 50 and 5
Debug: New command after evaluation is 55
Debug: Evaluating value of parsed string
Info: Output returned is 55
55
```

2) Sample input:-
```
java Test "mult(add(2, 2), div(9, 3))" "-- info" "-- error" "-- debug"
```

Sample Output:-
```
Info: String to be parsed after eliminating spaces is mult(add(2,2),div(9,3))
Debug: Reached a good evaluation point
Debug: Calling add with parameter 2 and 2
Debug: New command after evaluation is mult(4,div(9,3))
Debug: Reached a good evaluation point
Debug: Calling div with parameter 9 and 3
Debug: New command after evaluation is mult(4,3)
Debug: Reached a good evaluation point
Debug: Calling mult with parameter 4 and 3
Debug: New command after evaluation is 12
Debug: Evaluating value of parsed string
Info: Output returned is 12
12
```