1.

1. number of lines of code	760
2. number of lines in the shortest class	7
3. number of lines in the longest class	195
4. maximum number of lines in a method	41
5. minimum number of lines in method	3
6. maximum number of methods in a class	16
7. minimum number of methods in a class	1
8. number of imported Java libraries	27
9. number of packages	3
10. number of classes	9
11. number of abstract classes	1
12. number of interfaces	1
13. number of concrete classes	7
14. number of methods	69
15. number of static methods	1 (main)

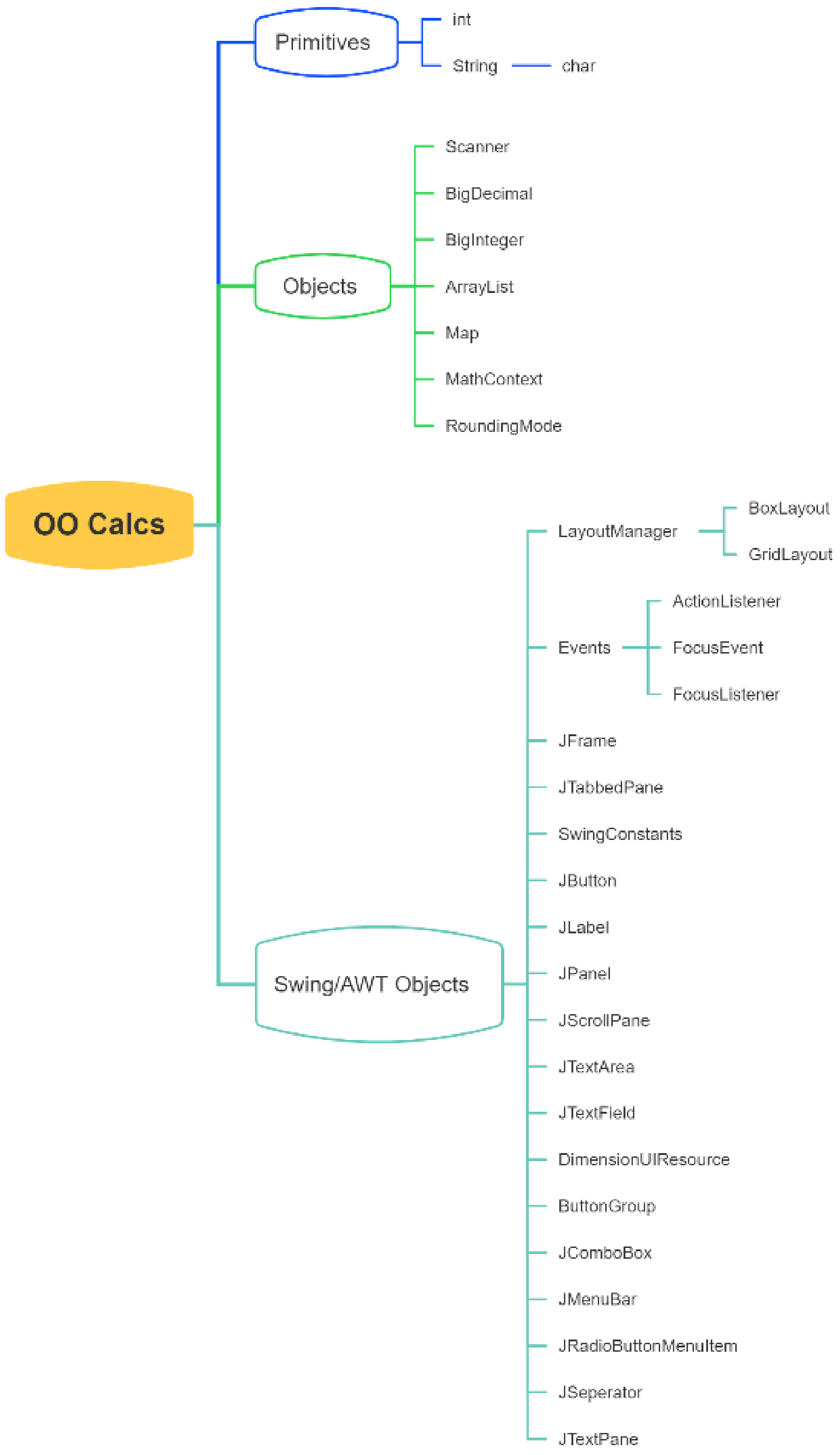
16. number of methods responsible for only input/output	System.out = 0
	JTextComponents = 7

17. number of methods responsible for some type of processing as well as input/output	System.out = 0
	JTextComponents = 16

18. number of methods responsible for some type of processing but no input/output	46
19. number of static variables	0
20. number of final static variables	15

2. https://gitmind.com/app/doc/7ff5886851

3.


OO Calcs
Primitives
int
String
char
Objects
Scanner
BigDecimal
BigInteger
ArrayList
Map
MathContext
RoundingMode
Swing/AWT Objects
LayoutManager
BoxLayout
GridLayout
Events
ActionListener
FocusEvent
FocusListener
JFrame
JTabbedPane
SwingConstants
JButton
JLabel
JPanel
JScrollPane
JTextArea
JTextField
DimensionUIResource
ButtonGroup
JComboBox
JMenuBar
JRadioButtonMenuItem
JSeperator
JTextPane

3.

Abstraction

-By inheritance, it is abstract enough that another programmer can add any of the calculators to their another JFrame. Another example is the two handlers for BigInteger and BigDecimal made doing different and new operations with two given values easy as it is already set up, so passing it 2 String of numbers and the operation will give an output.

Encapsulation

-The MainJFrame has the 3 JPanels of calculators and the 3 JPanels each have their input, output, and keys that do not interfere with each other. BigIntegerHandler and BigDecimalHandler each have 2 input fields and an output field they can independently manipulate.

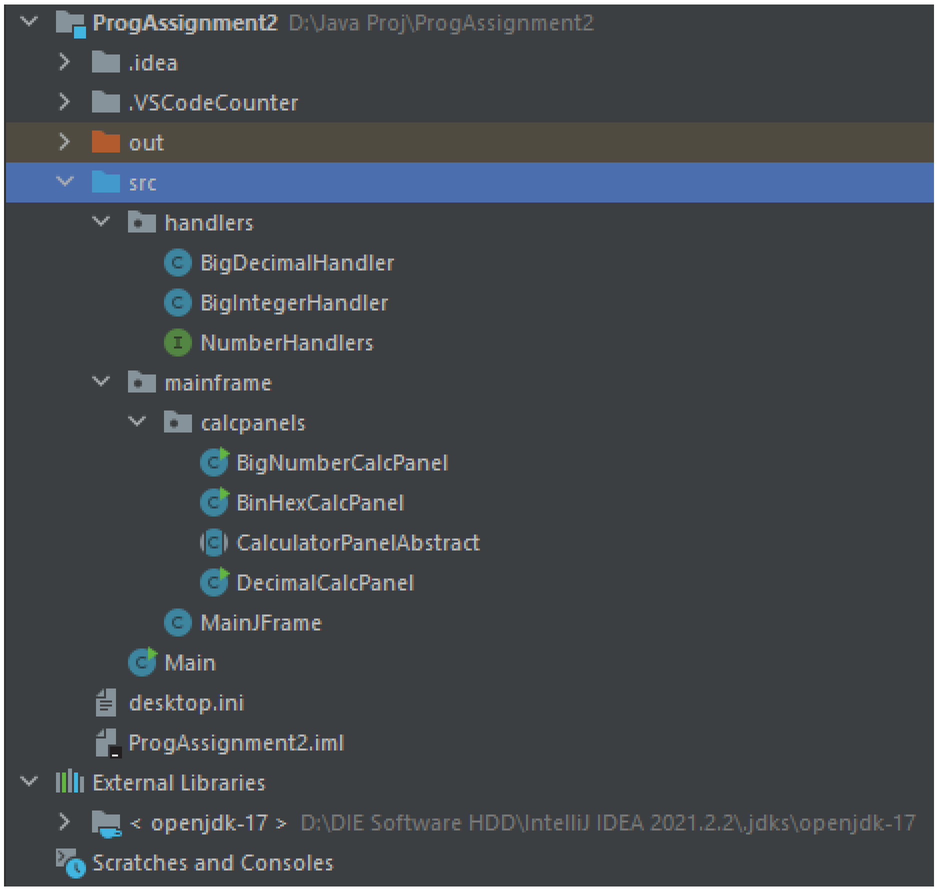
Inheritance

-All of the class that deals with Swing components are extended from the class with the closest functionality. CalculatorPanelAbstract also organizes the structure of the calculator panels that extends it. There's an input panel, output panel, and a numeric keys button panel.

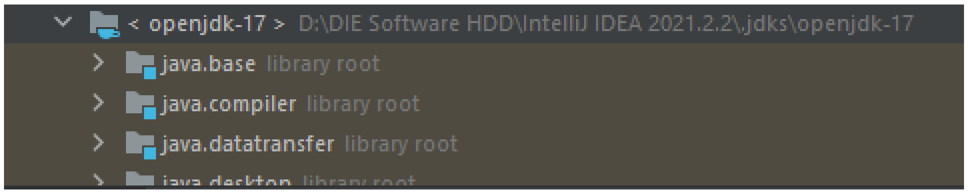
Polymorphism

-I used static polymorphism with various methods, one is computeOut() in BigDecimalHandler where you can specify the decimal precision in one with an int. I also used dynamic polymorphism, like getting the preferred dimension for each panel or overriding the methods that create the input, output, and key panels.

4. I didn't use any external libraries, it's just JDK 17.

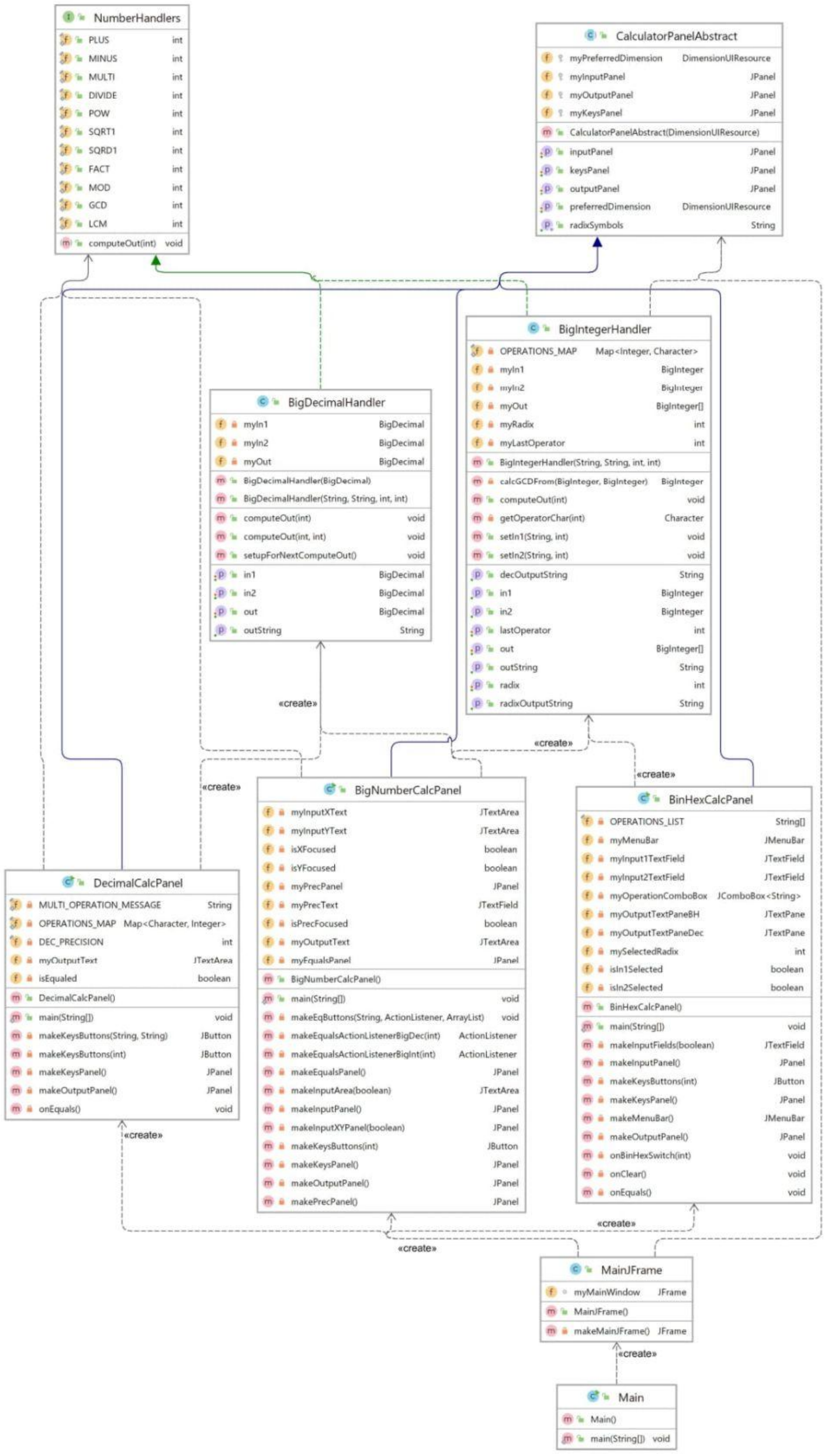


```
ProgAssignment2  D:\Java Proj\ProgAssignment2
  .idea
  .VSCodeCounter
  out
  src
    handlers
      BigDecimalHandler
      BigIntegerHandler
      NumberHandlers
    mainframe
      calcpanels
        BigNumberCalcPanel
        BinHexCalcPanel
        CalculatorPanelAbstract
        DecimalCalcPanel
      MainJFrame
    Main
  desktop.ini
  ProgAssignment2.iml
External Libraries
  < openjdk-17 >  D:\DIE Software HDD\IntelliJ IDEA 2021.2.2\.jdks\openjdk-17
Scratches and Consoles
```

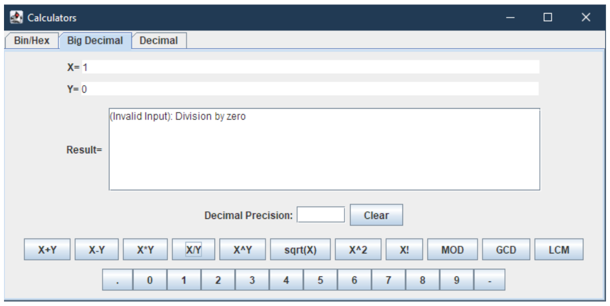


```
< openjdk-17 >  D:\DIE Software HDD\IntelliJ IDEA 2021.2.2\.jdks\openjdk-17
  java.base  library root
  java.compiler  library root
  java.datatransfer  library root
  java.desktop  library root
```

5. UML


NumberHandlers
PLUS int
MINUS int
MULTI int
DIVIDE int
POW int
SQRT1 int
SQRD1 int
FACT int
MOD int
GCD int
LCM int
computeOut(int) void
CalculatorPanelAbstract
myPreferredDimension DimensionUIResource
myInputPanel JPanel
myOutputPanel JPanel
myKeysPanel JPanel
CalculatorPanelAbstract(DimensionUIResource)
inputPanel JPanel
keysPanel JPanel
outputPanel JPanel
preferredDimension DimensionUIResource
radixSymbols String
BigIntegerHandler
OPERATIONS_MAP Map<Integer, Character>
myIn1 BigInteger
myIn2 BigInteger
myOut BigInteger[]
myRadix int
myLastOperator int
BigIntegerHandler(String, String, int, int)
calcGCDFrom(BigInteger, BigInteger) BigInteger
computeOut(int) void
getOperatorChar(int) Character
setIn1(String, int) void
setIn2(String, int) void
decOutputString String
in1 BigInteger
in2 BigInteger
lastOperator int
out BigInteger[]
outString String
radix int
radixOutputString String
BigDecimalHandler
myIn1 BigDecimal
myIn2 BigDecimal
myOut BigDecimal
BigDecimalHandler(BigDecimal)
BigDecimalHandler(String, String, int, int)
computeOut(int) void
computeOut(int, int) void
setupForNextComputeOut() void
in1 BigDecimal
in2 BigDecimal
out BigDecimal
outString String
«create»
«create»
«create»
«create»
DecimalCalcPanel
MULTI_OPERATION_MESSAGE String
OPERATIONS_MAP Map<Character, Integer>
DEC_PRECISION int
myOutputText JTextArea
isEqualed boolean
DecimalCalcPanel()
main(String[]) void
makeKeysButtons(String, String) JButton
makeKeysButtons(int) JButton
makeKeysPanel() JPanel
makeOutputPanel() JPanel
onEquals() void
BigNumberCalcPanel
myInputXText JTextArea
myInputYText JTextArea
isXFocused boolean
isYFocused boolean
myPrecPanel JPanel
myPrecText JTextField
isPrecFocused boolean
myOutputText JTextArea
myEqualsPanel JPanel
BigNumberCalcPanel()
main(String[]) void
makeEqButtons(String, ActionListener, ArrayList) void
makeEqualsActionListenerBigDec(int) ActionListener
makeEqualsActionListenerBigInt(int) ActionListener
makeEqualsPanel() JPanel
makeInputArea(boolean) JTextArea
makeInputPanel() JPanel
makeInputXYPanel(boolean) JPanel
makeKeysButtons(int) JButton
makeKeysPanel() JPanel
makeOutputPanel() JPanel
makePrecPanel() JPanel
BinHexCalcPanel
OPERATIONS_LIST String[]
myMenuBar JMenuBar
myInput1TextField JTextField
myInput2TextField JTextField
myOperationComboBox JComboBox<String>
myOutputTextPaneBH JTextPane
myOutputTextPaneDec JTextPane
mySelectedRadix int
isIn1Selected boolean
isIn2Selected boolean
BinHexCalcPanel()
main(String[]) void
makeInputFields(boolean) JTextField
makeInputPanel() JPanel
makeKeysButtons(int) JButton
makeKeysPanel() JPanel
makeMenuBar() JMenuBar
makeOutputPanel() JPanel
onBinHexSwitch(int) void
onClear() void
onEquals() void
«create»
«create»
«create»
MainJFrame
myMainWindow JFrame
MainJFrame()
makeMainJFrame() JFrame
«create»
Main
Main()
main(String[]) void

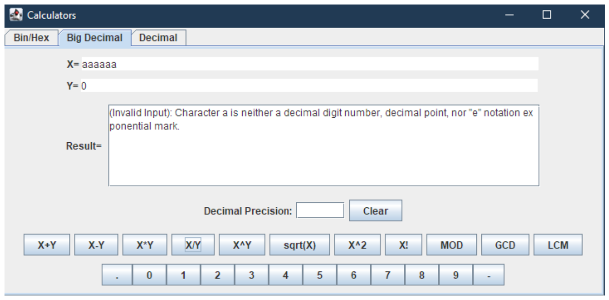
6.



X= 1

Y= 0

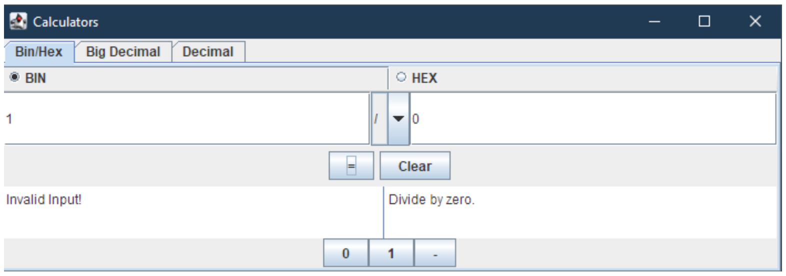
Result= (Invalid Input): Division by zero



X= aaaaaa

Y= 0

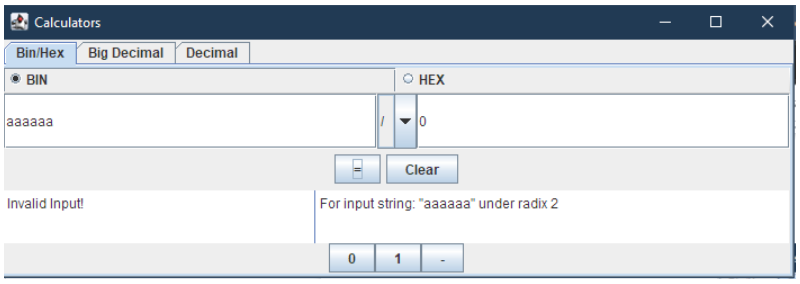
Result= (Invalid Input): Character a is neither a decimal digit number, decimal point, nor "e" notation exponential mark.



1 / 0

Invalid Input!

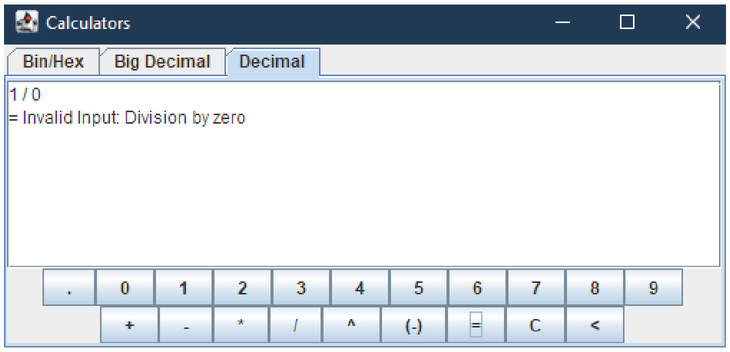
Divide by zero.



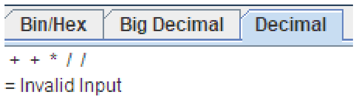
aaaaaa / 0

Invalid Input!

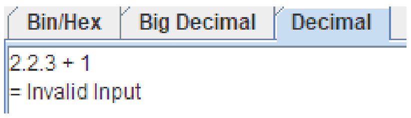
For input string: "aaaaaa" under radix 2



1 / 0
= Invalid Input: Division by zero



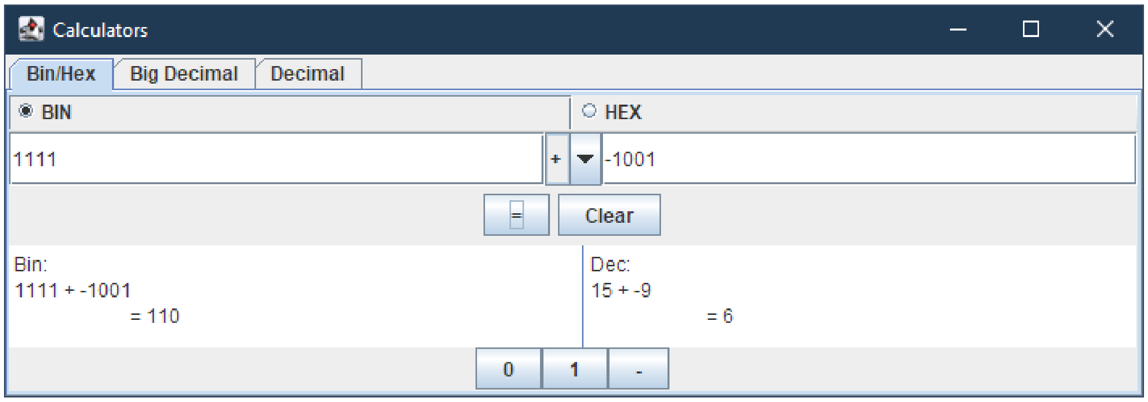
+ + * / /
= Invalid Input



2.2.3 + 1
= Invalid Input

7. Bin

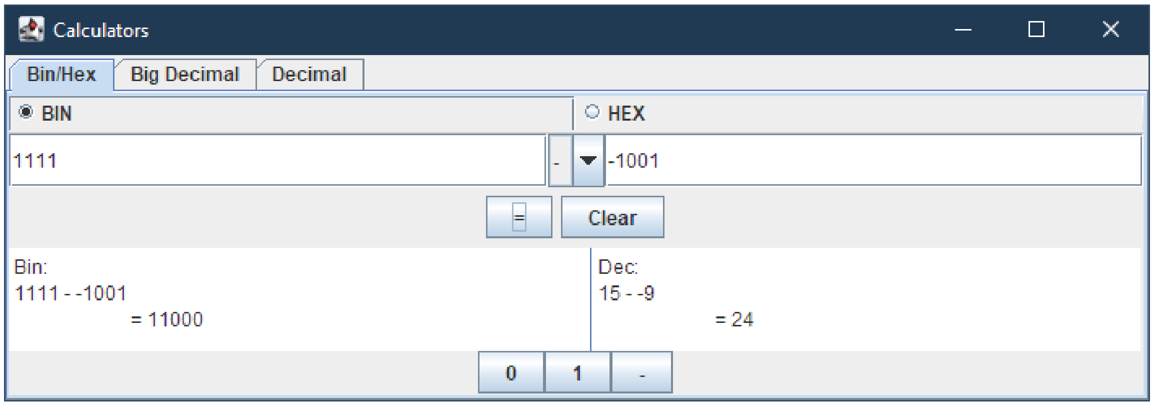
(The program is all GUI based, so I didn't use System.out)



1111 + -1001

```
Bin:
1111 + -1001
          = 110
```

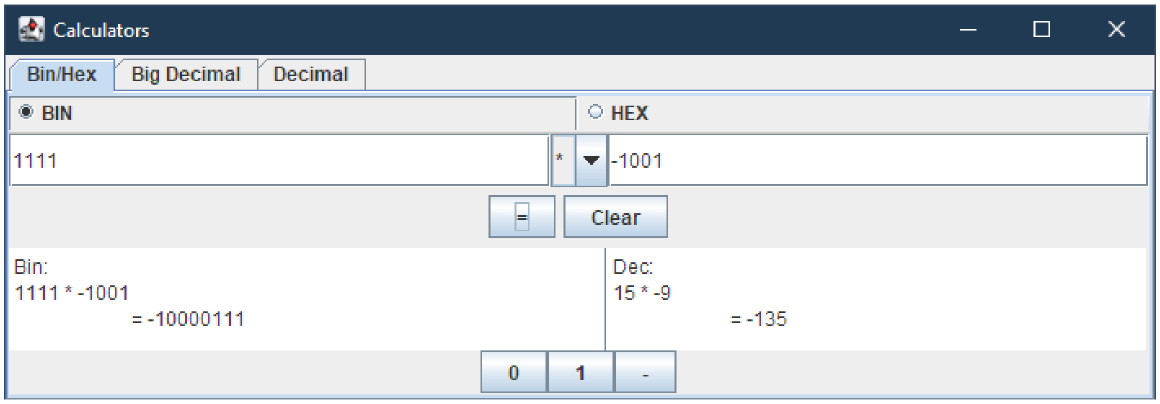
```
Dec:
15 + -9
          = 6
```



1111 - -1001

```
Bin:
1111 - -1001
          = 11000
```

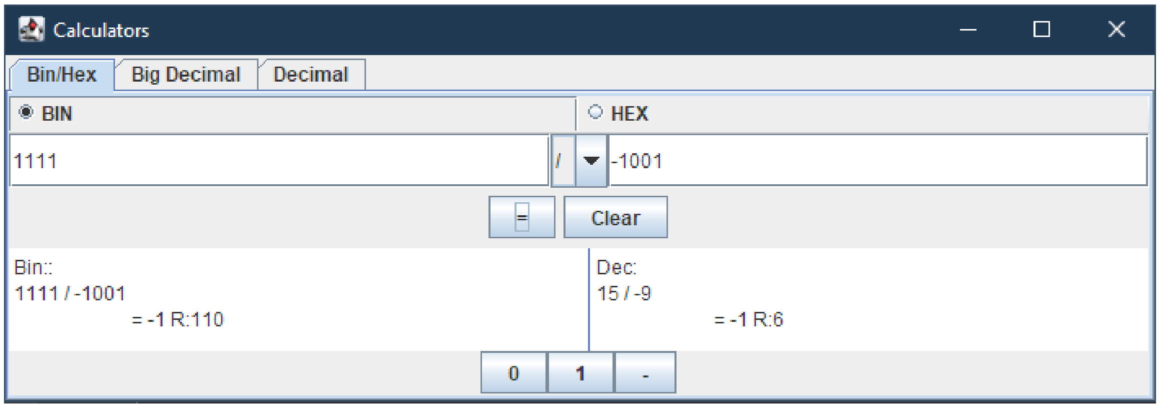
```
Dec:
15 - -9
          = 24
```



1111 * -1001

```
Bin:
1111 * -1001
          = -10000111
```

```
Dec:
15 * -9
          = -135
```

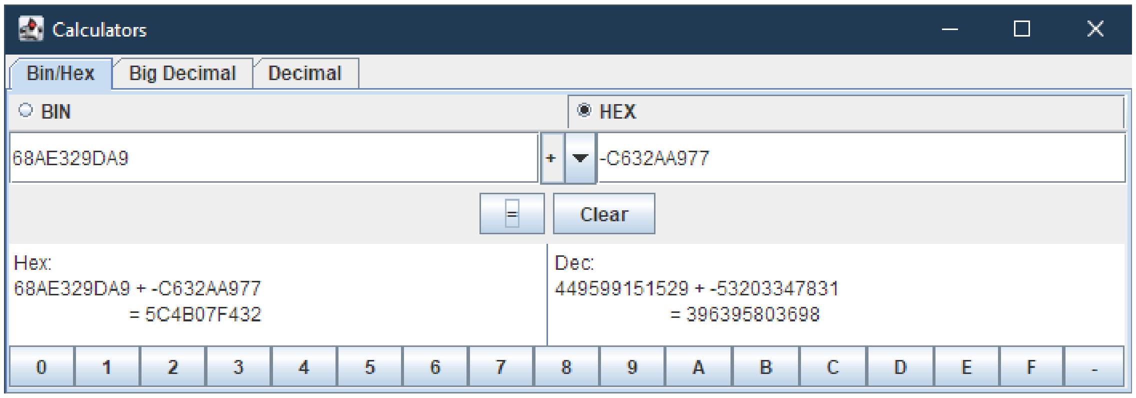


1111 / -1001

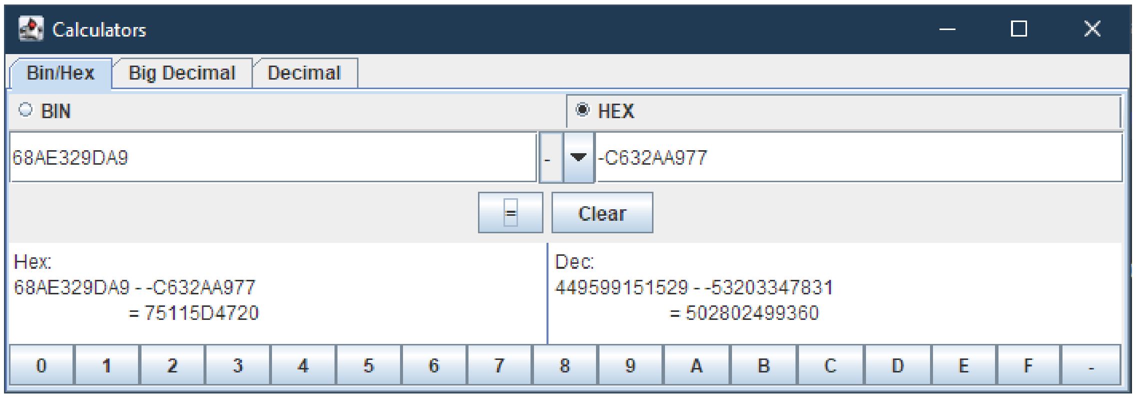
```
Bin::
1111 / -1001
          = -1 R:110
```

```
Dec:
15 / -9
          = -1 R:6
```

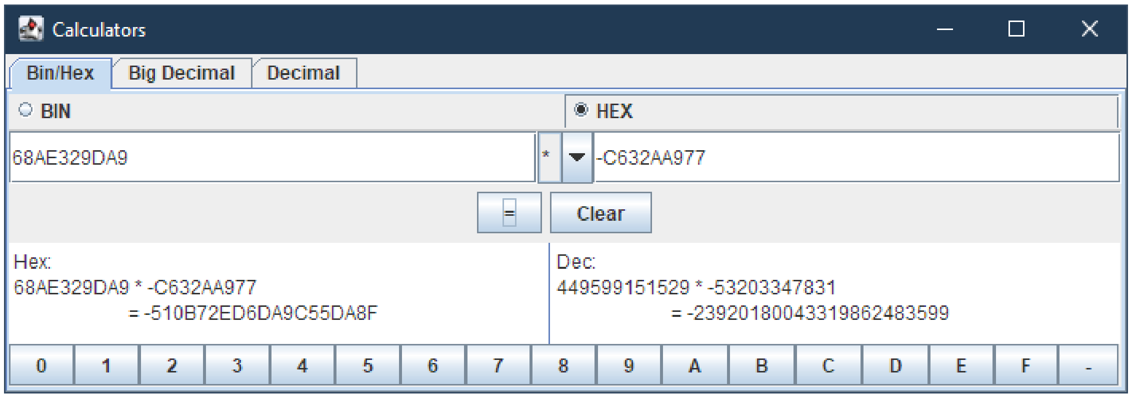
7. Hex



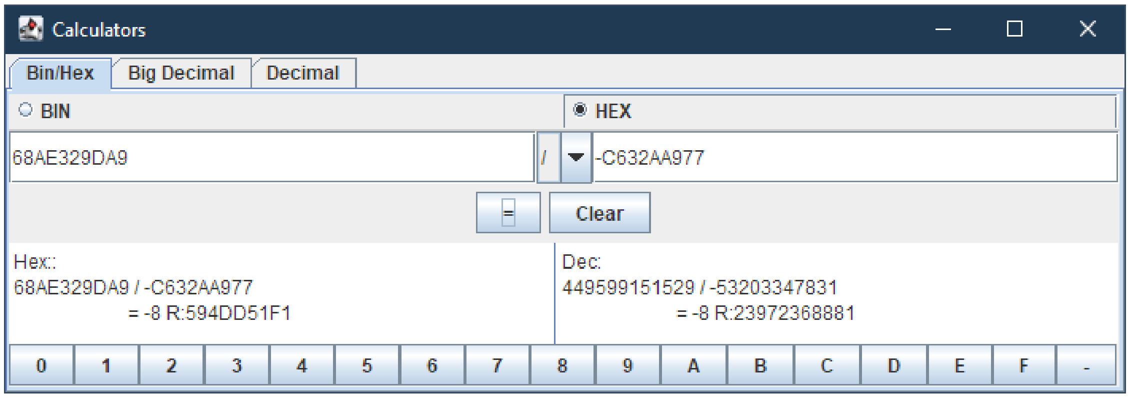
```
68AE329DA9  +  -C632AA977
Hex:
68AE329DA9 + -C632AA977
           = 5C4B07F432
Dec:
449599151529 + -53203347831
           = 396395803698
```



```
68AE329DA9  -  -C632AA977
Hex:
68AE329DA9 - -C632AA977
           = 75115D4720
Dec:
449599151529 - -53203347831
           = 502802499360
```



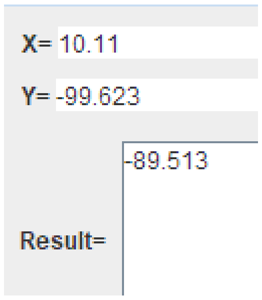
```
68AE329DA9  *  -C632AA977
Hex:
68AE329DA9 * -C632AA977
           = -510B72ED6DA9C55DA8F
Dec:
449599151529 * -53203347831
           = -23920180043319862483599
```



```
68AE329DA9  /  -C632AA977
Hex::
68AE329DA9 / -C632AA977
           = -8 R:594DD51F1
Dec:
449599151529 / -53203347831
           = -8 R:23972368881
```

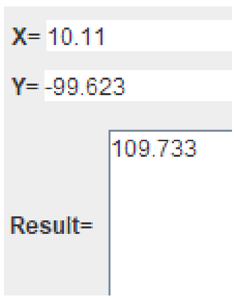
7. Big Number

-Addition



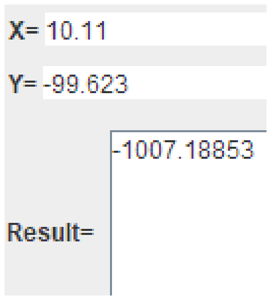
```
X= 10.11
Y= -99.623
Result= -89.513
```

-Subtraction



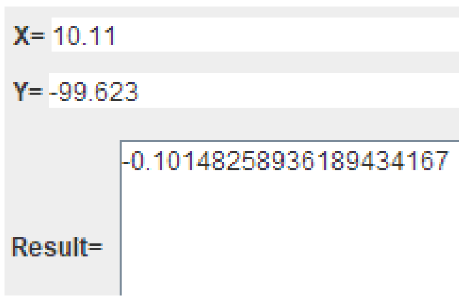
```
X= 10.11
Y= -99.623
Result= 109.733
```

-Multiply



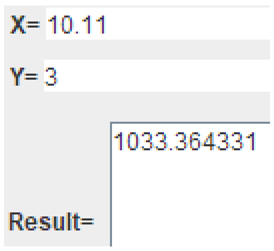
```
X= 10.11
Y= -99.623
Result= -1007.18853
```

-Divide



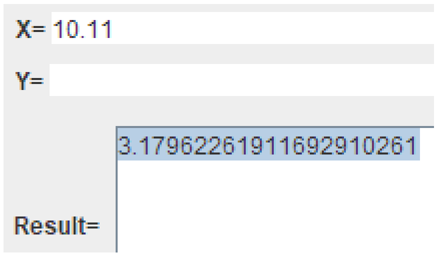
```
X= 10.11
Y= -99.623
Result= -0.10148258936189434167
```

-Pow



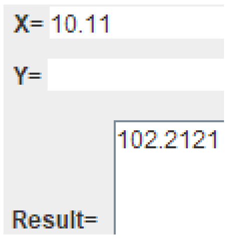
```
X= 10.11
Y= 3
Result= 1033.364331
```

-Sqrt(X)



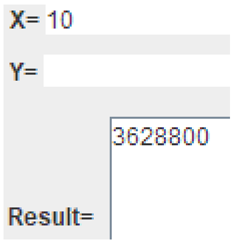
```
X= 10.11
Y=
Result= 3.17962261911692910261
```

-X^2



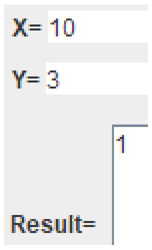
```
X= 10.11
Y=
Result= 102.2121
```

-Factorial



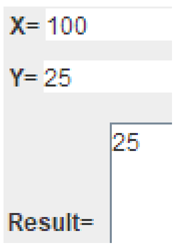
```
X= 10
Y=
Result= 3628800
```

-Modulo



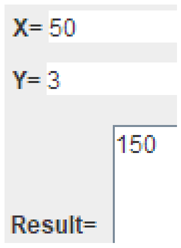
```
X= 10
Y= 3
Result= 1
```

-GCD



```
X= 100
Y= 25
Result= 25
```

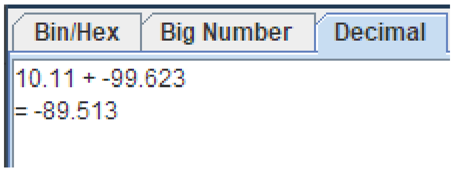
-LCM



```
X= 50
Y= 3
Result= 150
```

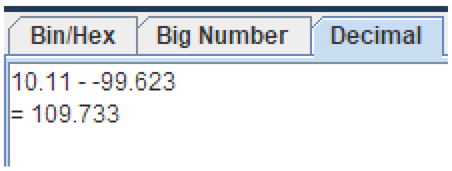
7. Decimal

-Addition



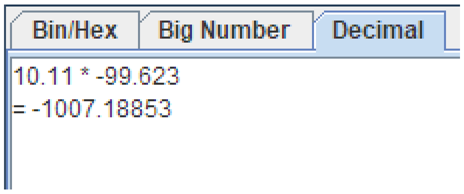
```
10.11 + -99.623
= -89.513
```

-Subtract



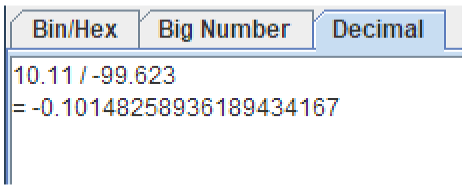
```
10.11 - -99.623
= 109.733
```

-Multiply



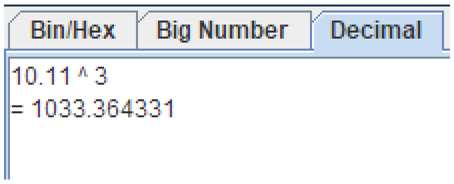
```
10.11 * -99.623
= -1007.18853
```

-Divide



```
10.11 / -99.623
= -0.10148258936189434167
```

-Pow



```
10.11 ^ 3
= 1033.364331
```

8a&b. This implementation is probably really sloppy. When the user gives their input, the calculator panels pass it to either of the two big number handlers to convert and compute. Whatever runtime exception gets thrown by the handlers (should be either NumberFormat or Arithmetic) is caught and the exception's message is displayed to the user if there is one. So while this catches both bad inputs and bad calculations, it perhaps displays the error messages without enough abstraction to the user. In retrospect, I should probably have made subclasses to NumberFormat & Arithmetic exceptions to specify which input field is wrong.

9.
-This assignment gave me experience using the style of components like containers, where one holds another, which holds another. The last time I did a serious assignment with GUI, I needed only one JPanel.
-This was the first time using GUI events in Java, which required making listeners using lambdas statements or overriding a method when passing an object into a parameter.
-This also is the first big assignment where I read user input from a GUI, which I used a Scanner in order to get the text entries.
-I've handled exceptions before, but I guess I've never displayed to the user exactly what is wrong. I don't think I did a good job here as I used the default message from Java exceptions to tell the user the problem. I should have carefully broken up the exception handling, catch an exception for each user input or something.
-The resulting work here is another one of the cases after the assignment where I'm not confident if the way I implemented it was the best. From looking through my code again to type this write-up, I'm unsatisfied with how I did a lot of parts of the implementation, especially the inheritance structure of the panels. It feels so spaghetti as I should have had the input, output, and numeric button panel also be a new class where getters and setters can allow each of them to manipulate each other. The text fields should have been a new class too, able to hold an "isFocus" variable where the buttons are further linked with the text fields instead of having the panels redirect the buttons to the text field. I never saw a really good example of GUI encapsulation before coding so I realized it only after writing it up.

10.
https://uw.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=f1ecc5f9-1d12-4c82-93d3-ade300340b89
(I think I rambled a bit when I explained #9. I was just trying to explain the last takeaway.)