Name	Shubhan Singh
UID no.	2022300118
Experiment No.	3

PROBLEM STATEMENT :	Implement Calculator (Menue driven - switch case) in Java.
THEORY:	How to take single character as input (We used this to take operator from input): In Java, you can take a single character as input using the **next().charAt(0)** method of the **Scanner** class. This method reads the next token from the input stream and returns the first character of that token. The **charAt(0)** method is used to extract the first character of the token. For example, suppose you have a **Scanner** object called **sc** that is reading input from the console. To read a single character from the input, you can use the following code: **char c = sc.next().charAt(0);**. This code reads the next token from the input and extracts the first character of that token, storing it in the **char** variable **c**. Note that this method assumes that the input is formatted correctly and that the next token contains at least one character. If the input is not formatted correctly or if the next token is empty, this method will throw an exception. Math.pow() and Math.round() : In Java, the **Math** class provides a set of useful mathematical functions, including **Math.pow()** and **Math.round()**. The **Math.pow()** method is used to calculate the power of a number. It takes two arguments, the base and the exponent, and returns the result of raising the base to the exponent. For example, **Math.pow(2, 3)** would return the value 8, which is 2 raised to the power of 3. The **Math.round()** method is used to round a floating-point number to the nearest integer. It takes a single argument, the number to be rounded, and returns the nearest integer. If the number is exactly halfway between two integers, it rounds to the nearest even integer. For example, **Math.round(2.5)** would return the value 3, and **Math.round(3.5)** would return the value 4. We calculated the logarithm by dividing the natural logarithm of the

argument by that of the base. We found the natural logarithm using the Taylor series expansion for $\ln(1-x)$

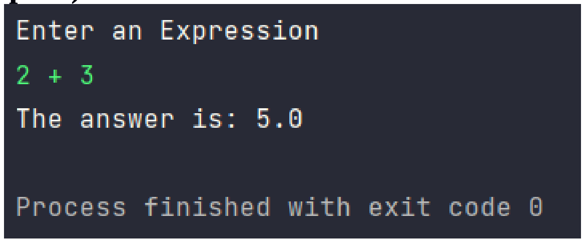
final keyword:
In Java, the **final** keyword can be used to define a constant variable or a method that cannot be overridden by its subclasses. If a variable is declared as **final**, its value cannot be changed once it is initialized. Similarly, if a method is declared as **final**, it cannot be overridden by any subclass.
The **final** keyword can also be used with classes to make them immutable, which means that their state cannot be changed once they are instantiated. A class declared as **final** cannot be subclassed, which helps in maintaining the integrity of the class.
Furthermore, the **final** keyword can also be used with method parameters to ensure that the value of the parameter is not changed within the method. This can be useful in scenarios where you want to ensure that a method does not modify the input parameters.

PROGRAM:

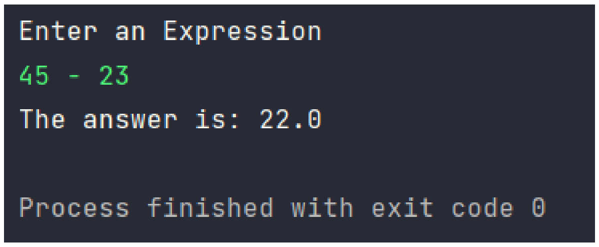
```java
import java.util.Scanner;
import java.lang.Math;

// Calci class to perform arithmetic and logarithmic
calculations
class Calci {
    float x; // first operand
    float y; // second operand
    char op; // operator(+,-,/,*,^,log,%)
    double ans; // result of calculation

    // constructor to initialize operands, operator and
perform calculation
    Calci(float x, float y, char op ){
        this.x = x;
        this.y = y;
        this.op = op;
        this.Calculate(); // perform calculation
    }
    // method to calculate the power of a number
    double pow(double base, int index){
        double ans=1;
        for(int i=0;i<index;i++){//The loop multiplies base
index number of times
            ans=ans*base;
        }
        return ans;
    }
    double calc_ln(double num){/*
         This method calculates the natural logarithm of a
number using the Taylor series
         expansion of the logarithm function. The loop
```

```
iterates 75 times(precision depends on number of times the
         loop is iterated), adding the i-th term of the
series to val,which is then multiplied by -1 to get the
natural
         logarithm of the number.
        */
        if(num==1){
            return 0;
        }
        if(num>1){

            return 1;
        }

        num=1-num;
        double val=0;
        for(int i=1;i<75;i++){
            val+=pow(num, i)/i;
        }
        val= -1*val;
        return val;
    }
    double calc_log(double num, double base){/*
         calculates the logarithm of a number with a given
base by dividing the
         natural logarithm of the number by the natural
logarithm of the base. It first converts the number and base
         to a number between 1 and 0, then calculates the
natural logarithms of the number and
         base, and finally uses those to calculate the
logarithm.
        */
        if(num==1){
            return 0;
        }
        if(num<=0 || base<=0){
            return 0;
        }

        double val;
        final double ln10=2.302585092994045684018;
        int powten=0;
        int powbase=0;

        // convert number and base to a number between 1 and
0
        while(num>1){
            powten++;
            num=num/10;
        }
        while(base>1){
            powbase++;
            base=base/10;
        }

        // calculate logarithm
        val=
(calc_ln(num)+powten*ln10)/(calc_ln(base)+powbase*ln10);
        return val;
    }
    void Calculate(){// method to perform the calculation
based on the operator
        switch (op) {
            case '+' -> ans = x + y;
            case '-' -> ans = x - y;
            case '*' -> ans = x * y;
            case '/' -> ans = x / y;
            case '^' -> ans = Math.pow(x, y);
            case '%' -> ans = x % y;
            case 'l' -> {
                ans = calc_log(x, y);
                int tempans = (int) Math.round(ans);
                // round off answer to 6 decimal places
                if (tempans - ans > 0) {
                    if (tempans - ans < 0.000001) {
                        ans = tempans;
                    }
                }
                else {
                    if (ans - tempans < 0.000001) {
                        ans = tempans;
                    }
                }
            }
        }
    }
}
// Calculator class to get user input and print result
class Calculator {
    public static void main(String[] args) {
        float x, y;
        char op;
        Scanner sc = new Scanner(System.in);

        // prompt user to enter an expression in the format x
op y
        System.out.println("Enter an Expression");
        x = sc.nextFloat();
        op = sc.next().charAt(0);
        y = sc.nextFloat();

        // create a new Calci object and print result of
calculation
        Calci calc = new Calci(x, y, op);
        System.out.println(“The answer is: “ + calc.ans);
        sc.close();
    }
}
```

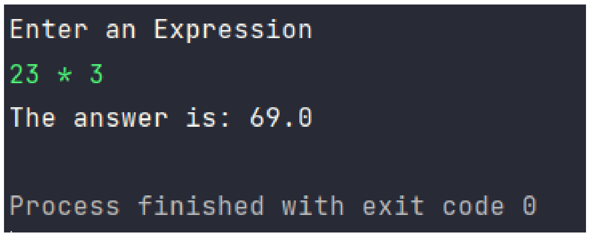
RESULT: (The program requires the operands and operator in the input to be separated by a space)



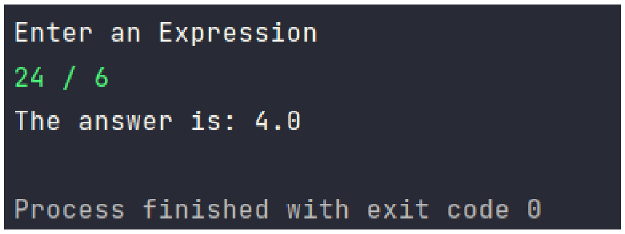
```
Enter an Expression
2 + 3
The answer is: 5.0

Process finished with exit code 0
```



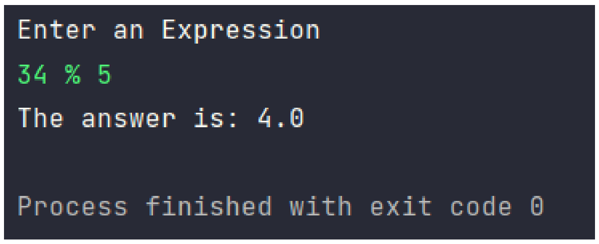
```
Enter an Expression
45 - 23
The answer is: 22.0

Process finished with exit code 0
```



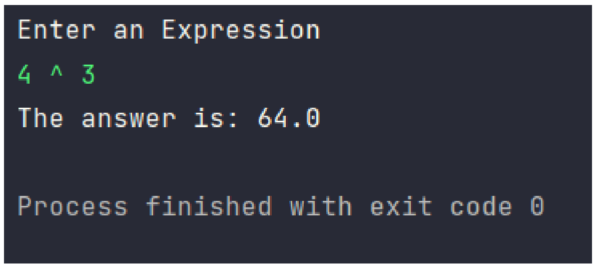
```
Enter an Expression
23 * 3
The answer is: 69.0

Process finished with exit code 0
```



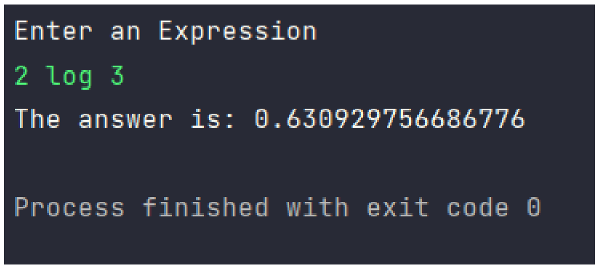
```
Enter an Expression
24 / 6
The answer is: 4.0

Process finished with exit code 0
```



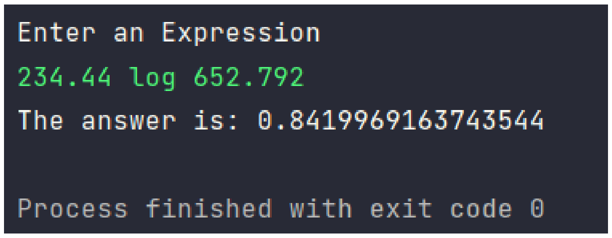
```
Enter an Expression
34 % 5
The answer is: 4.0

Process finished with exit code 0
```



```
Enter an Expression
4 ^ 3
The answer is: 64.0

Process finished with exit code 0
```



```
Enter an Expression
2 log 3
The answer is: 0.630929756686776

Process finished with exit code 0
```



```
Enter an Expression
234.44 log 652.792
The answer is: 0.8419969163743544

Process finished with exit code 0
```