Mini-project 2: BNF expression grammar

Read the program information, fill in the blanks and draw class diagram, interaction diagrams for the program.

You may draw several interaction diagrams. Note that the flow of them may illustrate for:

- the main method and similar scenarios calling other methods
- complicated methods of some classes (if any)

[Program Description]

The following are classes and test class for a simple expression.

(1) The expression grammar is defined in BNF as follows:

```
expression ::= variable | sequence sequence ::= expression
+ expression | expression - expression | expression *
expression | expression / expression
```

(2) Method `eval()` is to evaluate the value of an expression. All subclasses of expression need to implement this method.

(3) Method `setValue()` is to set an integer to a variable.

(4) Method `operate()` is to create a new expression by connecting two exist expression with an given operation (e.g. +, -, *, /).

[Program]

```
import java.io.*;
interface Expression {
    int eval();
}

class VarExp implements Expression {
    private int var;
    public VarExp() {};
    public void setValue(int n) {
        var = n;

    }
```

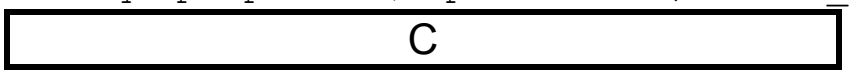
```
    public int eval() {
```


```
    }

}

class SeqExp implements Expression {
    private int op;
```


```
    public SeqExp(Expression e1, Expression e2, int a_op) {
        exp1 = e1;
        exp2 = e2;
        op = a_op;
    }
    public int eval() {
        switch (op) {
           case 0:
               return exp1.eval() + exp2.eval();
           case 1:
               return exp1.eval() - exp2.eval();
           case 2:
               return exp1.eval() * exp2.eval();
           case 3:
               return exp1.eval() / exp2.eval();
        }
        return 0;
    }
    public SeqExp operate(Expression e, int a_op) {
```




```
    }

}

public class TestExpression {
    public static void main(String args[]) {
        VarExp a = new VarExp();
        VarExp b = new VarExp();
        SeqExp sum = new SeqExp(a, b, 0);
        SeqExp diff = new SeqExp(a, b, 1);
        SeqExp mul = sum.operate(diff,2);
        a.setValue(3);
```

```
        b.setValue(7);
        System.out.print(mul.eval());
    }
}
```