**step 1**

package com.nsw.backend.service;  
  
import java.util.List;  
  
public interface Calculation<I, P> {  
 I calculate(boolean ignoreError, I inputToUpdate, List<Formula> formulas, P... params);  
  
 class Formula {  
  
 private int index;  
 private String value;  
  
 public int getIndex() {  
 return index;  
 }  
  
 public void setIndex(int index) {  
 this.index = index;  
 }  
  
 public String getValue() {  
 return value;  
 }  
  
 public void setValue(String value) {  
 this.value = value;  
 }  
  
 public Formula(int indexInSourceList, String formulaWithIndex) {  
 this.index = indexInSourceList;  
 this.value = formulaWithIndex;  
 }  
 }  
  
 interface CalculationParams {  
 Double getValue();  
 void setValue(Double value);  
 }  
  
 class Params implements CalculationParams {  
 private Double value;  
  
 public Params(Double value) {  
 this.value = value;  
 }  
  
 @Override  
 public Double getValue() {  
 return this.value;  
 }  
  
 @Override  
 public void setValue(Double value) {  
 this.value = value;  
 }  
 }  
  
 String evaluate(boolean ignoreError, String formulaValue, P... params);  
}

**step 2**

import com.nsw.backend.service.Calculation;  
import com.udojava.evalex.Expression;  
import org.apache.commons.lang3.math.NumberUtils;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
import java.math.MathContext;  
import java.util.List;  
import java.util.Objects;  
  
*/\*\*  
 \* Calculation-helper  
 \* Use for cases of recalculating a list base on formulas and params  
 \*/*public abstract class CalculationImpl<Input extends Calculation.CalculationParams, Param extends Calculation.CalculationParams> implements Calculation<List<Input>, List<Param>> {  
 private final Logger log = LoggerFactory.*getLogger*(CalculationImpl.class);  
  
 */\*\*  
 \* Update inputs by evaluating formulas.  
 \*  
 \** ***@param*** *params: a list of params that will be used when evaluating formulas  
 \** ***@throws*** *IndexOutOfBoundsException: possibly throws when @Param ignoreError is false and params is not match with formulas  
 \** ***@Param*** *ignoreError is true then exception in matching params with formula will take zero-0 instead of throwing IndexOutOfBoundException  
 \** ***@Param*** *inputs: the list that is updated by events  
 \** ***@Param*** *formulas: a list of formula containing rules to use params  
 \*/* @SafeVarargs  
 @Override  
 public final List<Input> calculate(boolean ignoreError, List<Input> inputs, List<Calculation.Formula> formulas, List<Param>... params) {  
 if (Objects.*isNull*(inputs) || inputs.isEmpty()) return inputs;  
 for (Calculation.Formula foo : formulas) {  
 if (Objects.*isNull*(foo)) continue;  
 String formulaValue = foo.getValue();  
 int indexInSourceList = foo.getIndex();  
  
 String roughValue = evaluate(ignoreError, formulaValue, params);  
 double value;  
 try {  
 value = new Expression(roughValue, MathContext.*DECIMAL128*).eval().doubleValue();  
 } catch (Exception e) {  
 log.info("Cannot evaluating: " + formulaValue + " = " + roughValue + ", due to ", e);  
 value = 0D;  
 }  
  
 Input input = inputs.get(indexInSourceList);  
 input.setValue(value);  
 inputs.set(indexInSourceList, input);  
 }  
 return inputs;  
 }  
  
 */\*\*  
 \* Evaluate a String formula by values of params in @param:params  
 \*  
 \** ***@Param*** *formula is the string containing params and the index of subParam in one param  
 \*/* @SafeVarargs  
 @Override  
 public final String evaluate(boolean ignoreError, String formula, List<Param>... params) {  
 formula = formula.replaceAll(" ", "");  
 StringBuilder result = new StringBuilder();  
 char[] roughValueInArray = formula.toCharArray();  
 int length = roughValueInArray.length;  
 int i = 0;  
 while (i < length) {  
 char thisChar = roughValueInArray[i];  
 int j = i + 1;  
 if (j == length) {  
 result.append(thisChar);  
 return result.toString();  
 }  
 char nextChar = roughValueInArray[j];  
  
 if (Character.*isLetter*(thisChar) && Character.*isDigit*(nextChar)) {  
 for (; j < length; j++) {  
 if (!Character.*isDigit*(roughValueInArray[j])) {  
 break;  
 }  
 }  
 int paramsIndex = getParamIndex(thisChar);  
 int paramsIndexIndex = Integer.*parseInt*(formula.substring(i + 1, j));  
 double value;  
 try {  
 value = NumberUtils.*createDouble*(params[paramsIndex].get(paramsIndexIndex).getValue().toString());  
 } catch (Exception ex) {  
 if (ignoreError) {  
 value = 0D;  
 } else {  
 log.error("Param: [" + thisChar + paramsIndexIndex + "] can not be evaluated-(" + formula + ")", ex);  
 throw ex;  
 }  
 }  
 result.append(value);  
 i = j - 1;  
 } else {  
 result.append(thisChar);  
 }  
  
 i++;  
 }  
 return result.toString();  
 }  
  
 private int getParamIndex(char p) {  
 p = Character.*toLowerCase*(p);  
 return (int) p - 97;  
 }  
}

ex: calculate(true, dataKH, formulas1, dataKH, dataTH);