Hacer la lógica de operaciones

Al hacer click sobre el botón de = se deberá hacer la lógica de las operaciones y mostrarla en el Textview secundario.

```
@Click
void b_equal() {
    String toRemoveZeros = tvOperations.getText().toString();
    String writeInTvResult = "", finalResult;

    if ( isThereNumbers(toRemoveZeros) ) {
        writeInTvResult = removeInitialZeros(toRemoveZeros);
        writeInTvResult = removeLastDecimalsZero(writeInTvResult);
    }
    finalResult = executeOperations(writeInTvResult);

    tv_results.setText(finalResult);
}

private boolean isThereNumbers(String stringToEvaluate) {
    return stringToEvaluate != null && ( stringToEvaluate.contains("
            stringToEvaluate.contains("1") ||
            stringToEvaluate.contains("2") ||
            stringToEvaluate.contains("3") ||
            stringToEvaluate.contains("4") ||
            stringToEvaluate.contains("5") ||
            stringToEvaluate.contains("6") ||
            stringToEvaluate.contains("7") ||
            stringToEvaluate.contains("8") ||
            stringToEvaluate.contains("9") );
}

private String removeInitialZeros(String toRemove) {
    final char ZERO = '0';
    final char DOT = '.';
    String toReturn;

    if ( (toRemove != null) &&
            (toRemove.length() > 0) &&
            (ZERO == toRemove.charAt(0)) ) {

        toReturn = removeInitialZeros(toRemove.substring(1));
    } else {
        if ( (toRemove == null) || (toRemove.isEmpty()) ) {
            toReturn = String.valueOf(ZERO);
        } else {
            toReturn = toRemove;
        }
        if ((DOT == toReturn.charAt(0))) {
```

```
            toReturn = "0" + toReturn;
        }
    }

    return toReturn;
}

private String removeLastDecimalsZero(String toRemove) {
    final String ZERO = "0";
    final String DOT = ".";
    String toReturn;

    if (toRemove.contains(DOT)) {
        String[] splittedDecimal = toRemove.split("\\.");
        String decimals = splittedDecimal[1];
        String revertedDecimals = reverseString(decimals);
        String withoutZeros = removeInitialZeros(revertedDecimals);
        if (ZERO.equals(withoutZeros)) {
            toReturn = splittedDecimal[0];
        } else {
            toReturn = splittedDecimal[0] + DOT + reverseString(with
        }
    } else {
        toReturn = toRemove;
    }

    return toReturn;
}

private String reverseString(String value) {
    return new StringBuilder(value).reverse().toString();
}

private String executeOperations(String operations) {
    String finalResult = "";
    List<String> recoveredOperations = recoverOperations(operations)
    recoveredOperations = removeOperatorsFromStart(recoveredOperatio
    Collections.reverse(recoveredOperations);
    recoveredOperations = removeOperatorsFromStart(recoveredOperatio
    Collections.reverse(recoveredOperations);

    if (!recoveredOperations.isEmpty()) {
        BigDecimal firstValue = new BigDecimal( recoveredOperations.
        recoveredOperations.remove(0);
        for (int c = 0; c < recoveredOperations.size(); c = c + 2) {
            try {
                firstValue = operate(firstValue,
                        new BigDecimal( recoveredOperations.get(c+1)
                        recoveredOperations.get(c));
            } catch (NumberFormatException | ArithmeticException e)
```

```
                e.printStackTrace();
                tvOperations.setText("");
            }
        }
        finalResult = removeLastDecimalsZero( firstValue.toString()
    }

    return finalResult;
}

private List<String> recoverOperations(String operations) {
    final String DOT = ".";
    List<String> values = new ArrayList<>();
    StringBuilder tmpNumber = new StringBuilder();

    for (int c = 0; c < operations.length(); c++) {
        String anValueFromOperations = String.valueOf(operations.cha
        if ( isThereNumbers(anValueFromOperations) || DOT.equals(anV
            tmpNumber.append(anValueFromOperations);
        } else {
            values.add( tmpNumber.toString() );
            values.add(anValueFromOperations);
            tmpNumber.setLength(0);
        }
    }
    values.add( tmpNumber.toString() );

    return values;
}

private List<String> removeOperatorsFromStart(List<String> in) {
    if (in != null && !in.isEmpty())
        if ( !isNumeric( in.get(0) ) ) {
            String a = in.remove(0);
            removeOperatorsFromStart(in);
        }

    return in;
}

private BigDecimal operate(BigDecimal firstOperator, BigDecimal seco
    BigDecimal toReturn = BigDecimal.valueOf(0);

    switch (operation) {
        case "/":
            final int SCALE = 5;
            toReturn = firstOperator.divide(secondOperator, SCALE, B
            break;
        case "*":
            toReturn = firstOperator.multiply(secondOperator);
```

```
            break;
        case "-":
            toReturn = firstOperator.subtract(secondOperator);
            break;
        case "+":
            toReturn = firstOperator.add(secondOperator);
            break;
    }

    return toReturn;
}

private boolean isNumeric(String strNum) {
    try {
        Double.parseDouble(strNum);
        return true;
    } catch (NumberFormatException | NullPointerException nfe) {
        return false;
    }
}
```