Jason Boccuti
CISC 213
Lab 1

Pseudocode

Time complexity for all 5 should be O(1), constant time, as there are no loops in the functions and just a few conditionals.

```
function logicalImplication(p, q)
      if p is true and q is false
            return false
      else
            return true
```

```
function logicalEquality(p, q)
      if p and q are both true
            return true
      else if p and q are both false
            return true
      else
            return false
```

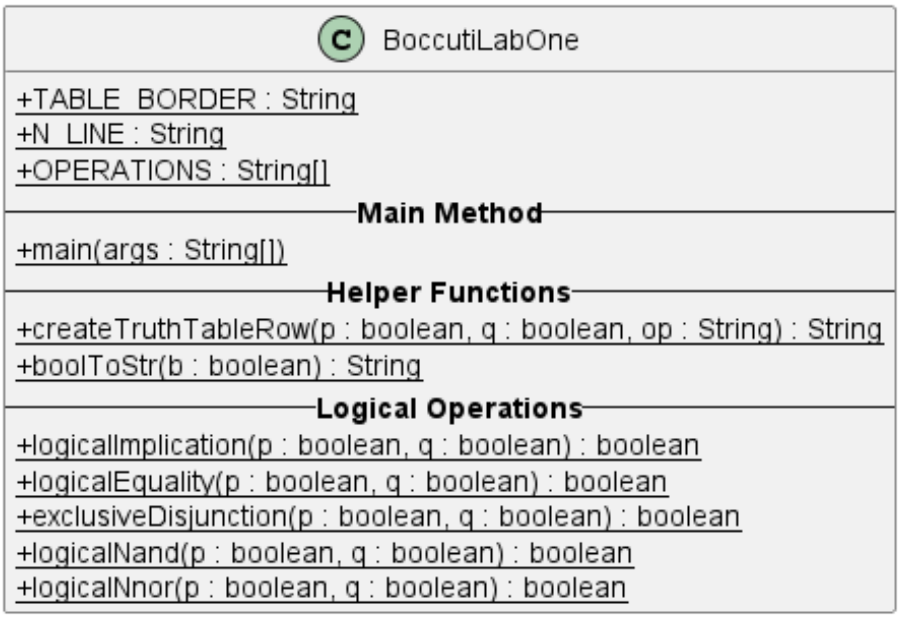
```
function exclusiveDisjunction(p, q)
      if p is true and q is false
            return true
      else if p is false and q is true
            return true
      else
            return false
```

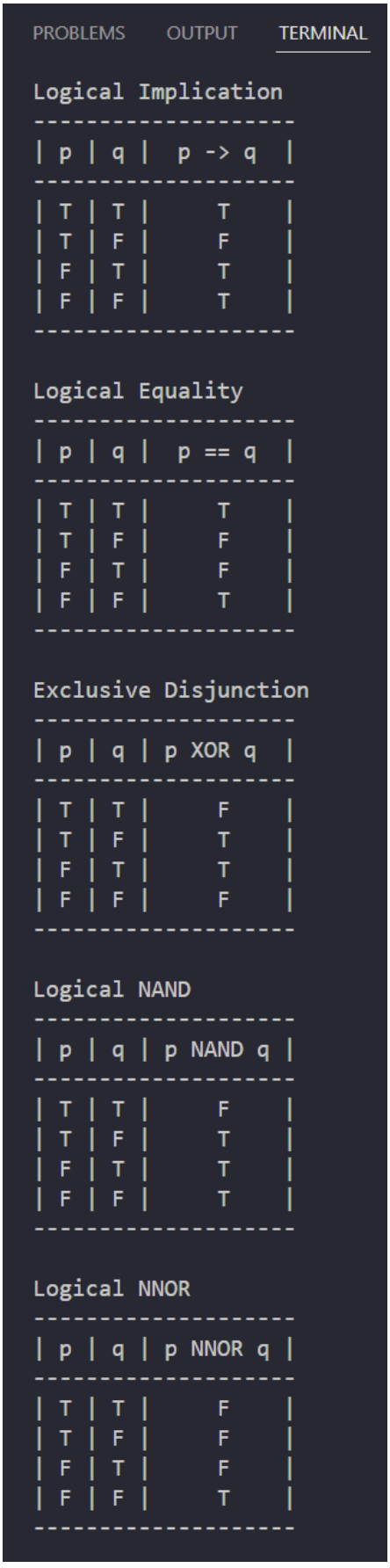
```
function logicalNAND(p, q)
      if p and q are both true
            return false
      else
            return true
```

```
function logicalNNOR(p, q)
      if p and q are both false
            return true
      else return false
```

Jason Boccuti
CISC 213
Lab 1

UML Class Diagram + Console Output





Jason Boccuti
CISC 213
Lab 1

Source Code

Source file also uploaded on Canvas

```java
package dev.boccuti.www.cisc213.lab1;

/**
 * Implementation of Lab #1 for CISC 213 that completes examples and builds
 * truth tables for 5 Boolean Operations: Logical Implication, Logical Equality,
 * Exclusive Disjunction, Logical NAND, and Logical NNOR. Purposefully avoids
 * using
 * any very clever shortcuts for the operations for the sake of academic
 * clarity.
 * For the same reason, explicit equality checks of booleans are used instead of
 * things
 * like !b for false or similar.
 *
 * @author Jason Boccuti | jason@boccuti.dev
 */
public class BoccutiLabOne {

    // Constants used to decrease copy-pasta code and reduce errors
    public static final String TABLE_BORDER = "--------------------";
    public static final String N_LINE = "\n";
    public static final String[] OPERATIONS = {
            "Logical Implication",
            "Logical Equality",
            "Exclusive Disjunction",
            "Logical NAND",
            "Logical NNOR"
    };

    /**
     * Main method of the lab that produces the truth table output strings
     * to console.
     *
     * @param args command line arguments are not used in this program
     */
    public static void main(String args[]) {

        // Variables for simplifying print statement building
        String operation = OPERATIONS[0];
        String tableStart = N_LINE + TABLE_BORDER + N_LINE;
```

Jason Boccuti
CISC 213
Lab 1

```
        String tableHeader = "| p | q |  p -> q  |";
        String divider = N_LINE + TABLE_BORDER;

        // Logical Implication Truth Table
        System.out.println(operation
                + tableStart
                + tableHeader
                + divider
                + createTruthTableRow(true, true, operation)
                + createTruthTableRow(true, false, operation)
                + createTruthTableRow(false, true, operation)
                + createTruthTableRow(false, false, operation)
                + divider);

        // Logical Equality Truth Table
        operation = OPERATIONS[1];
        tableHeader = "| p | q |  p == q  |";
        System.out.println(N_LINE
                + operation
                + tableStart
                + tableHeader
                + divider
                + createTruthTableRow(true, true, operation)
                + createTruthTableRow(true, false, operation)
                + createTruthTableRow(false, true, operation)
                + createTruthTableRow(false, false, operation)
                + divider);

        // Exclusive Disjunction Truth Table
        operation = OPERATIONS[2];
        tableHeader = "| p | q | p XOR q  |";
        System.out.println(N_LINE
                + operation
                + tableStart
                + tableHeader
                + divider
                + createTruthTableRow(true, true, operation)
                + createTruthTableRow(true, false, operation)
                + createTruthTableRow(false, true, operation)
                + createTruthTableRow(false, false, operation)
                + divider);

        // Logical NAND Truth Table
```

```
        operation = OPERATIONS[3];
        tableHeader = "| p | q | p NAND q |";
        System.out.println(N_LINE
                + operation
                + tableStart
                + tableHeader
                + divider
                + createTruthTableRow(true, true, operation)
                + createTruthTableRow(true, false, operation)
                + createTruthTableRow(false, true, operation)
                + createTruthTableRow(false, false, operation)
                + divider);

        // Logical NNOR Truth Table
        operation = OPERATIONS[4];
        tableHeader = "| p | q | p NNOR q |";
        System.out.println(N_LINE
                + operation
                + tableStart
                + tableHeader
                + divider
                + createTruthTableRow(true, true, operation)
                + createTruthTableRow(true, false, operation)
                + createTruthTableRow(false, true, operation)
                + createTruthTableRow(false, false, operation)
                + divider);
    }

    /**
     * Helper function for generating the truth table rows for the given
     * p, q, and operation type values.
     *
     * @param p  the given p value
     * @param q  the given q value
     * @param op the operation name string
     * @return the string for the row of the truth table
     */
    public static String createTruthTableRow(boolean p, boolean q, String op) {

        boolean result;

        /*
         * Check the operation passed in, then complete the correct operation
```

Jason Boccuti
CISC 213
Lab 1

```
         * using the given p and q values and store in the result to be concatenated
         * with the return string
         */
        if (op.equals(OPERATIONS[0])) {
            result = logicalImplication(p, q);
        } else if (op.equals(OPERATIONS[1])) {
            result = logicalEquality(p, q);
        } else if (op.equals(OPERATIONS[2])) {
            result = exclusiveDisjunction(p, q);
        } else if (op.equals(OPERATIONS[3])) {
            result = logicalNand(p, q);
        } else if (op.equals(OPERATIONS[4])) {
            result = logicalNnor(p, q);
        } else {
            result = false;
        }

        // Build the return string and prepend with a newline character so it formats
        // correctly
        return N_LINE + "| " + boolToStr(p) + " | " + boolToStr(q) + " |     " +
boolToStr(result) + "    |";
    }

    /**
     * Helper function to quickly translate boolean values to their respective
     * 1 letter string values (T or F) for easy use in building the truth table
     * strings.
     *
     * @param b the boolean to convert
     * @return the 1 digit string conversion
     */
    public static String boolToStr(boolean b) {

        if (b) {
            return "T";
        }

        return "F";
    }

    /**
     * The implementation of the Logical Implication operation using simple
     * conditional checks for the important case.
```

Jason Boccuti
CISC 213
Lab 1

```
     *
     * @param p the given p value
     * @param q the given q value
     * @return the correct boolean result
     */
    public static boolean logicalImplication(boolean p, boolean q) {
        // Logical Implication is only false in this case, so all others are true
        if (p == true && q == false) {
            return false;
        } else {
            return true;
        }
    }

    /**
     * The implementation of the Logical Equality operation using simple
     * conditional checks for the two important cases.
     *
     * @param p the given p value
     * @param q the given q value
     * @return the correct boolean result
     */
    public static boolean logicalEquality(boolean p, boolean q) {
        /*
         * Logical Equality is only true when p and q are the same. This could
         * be simplified to if(p == q) as the only true case, but for clarity included
         * the two actual truth table cases.
         */
        if (p == true && q == true) {
            return true;
        } else if (p == false && q == false) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * The implementation of the Exclusive Disjunction operation using simple
     * conditional checks for the two important cases.
     *
     * @param p the given p value
     * @param q the given q value
```

Jason Boccuti
CISC 213
Lab 1

```
     * @return the correct boolean result
     */
    public static boolean exclusiveDisjunction(boolean p, boolean q) {
        // Exclusive Disjunction is only true IFF one value is true
        if (p == true && q == false) {
            return true;
        } else if (p == false && q == true) {
            return true;
        } else {
            return false;
        }
    }

    /**
     * The implementation of the Logical NAND operation using simple
     * conditional checks for the one important case.
     *
     * @param p the given p value
     * @param q the given q value
     * @return the correct boolean result
     */
    public static boolean logicalNand(boolean p, boolean q) {
        // Logical NAND is only false when both values are true
        if (p == true && q == true) {
            return false;
        }
        return true;
    }

    /**
     * The implementation of the Logical NNOR operation using simple
     * conditional checks for the one important case.
     *
     * @param p the given p value
     * @param q the given q value
     * @return the correct boolean result
     */
    public static boolean logicalNnor(boolean p, boolean q) {
        // Logical NNOR is only true when both values are false
        if (p == false && q == false) {
            return true;
        }
        return false;
```

Jason Boccuti
CISC 213
Lab 1

```
    }
}
```