Sohan Chatterjee

Lab 11 Complexity Analysis

SSW 315

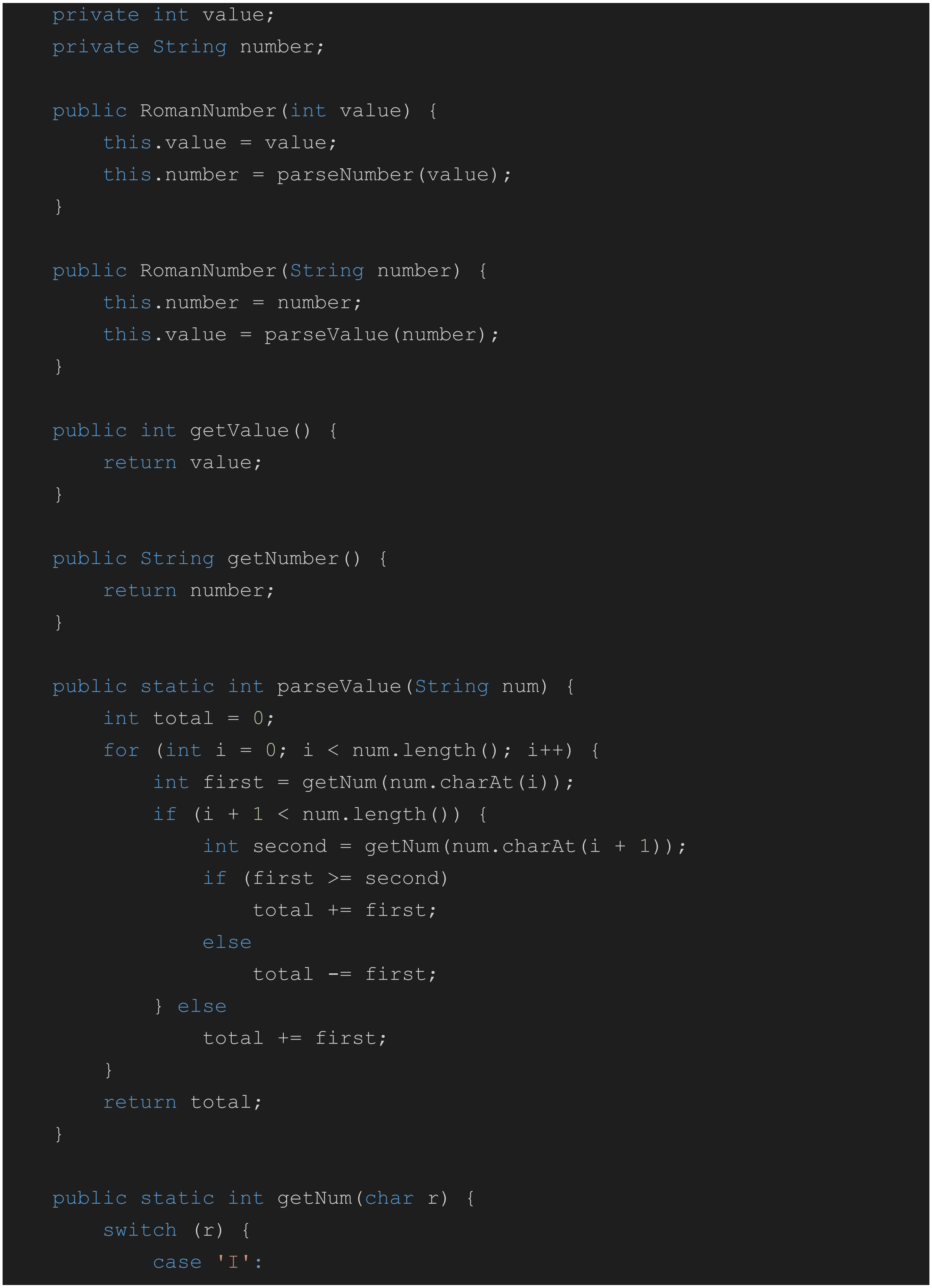
14 November 2022

Complexity Analysis #3 - Roman Number

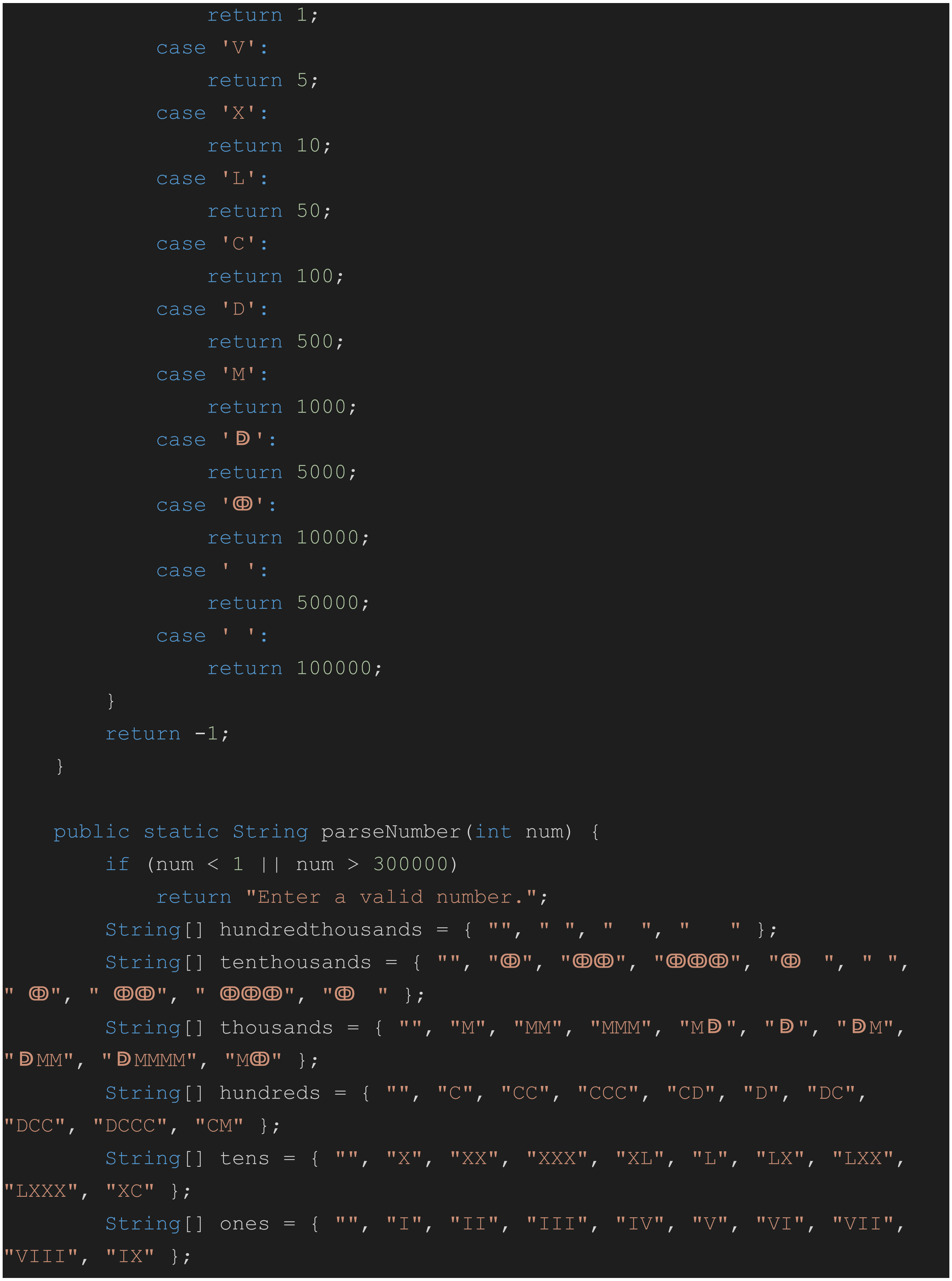
Method	Worst Case	Storage Complexity	Justification for non-constant complexities
RomanNumber(int value)	none	O(1)	
RomanNumber(String number)	n=num.length()	O(n)	The constructor calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
getValue()	none	O(1)	
getNumber()	none	O(1)	
parseValue(String num)	n=num.length()	O(n)	The function runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
getNum(char r)	none	O(1)	
parseNumber(int num)	none	O(1)	
equals(String num)	none	O(1)	Uses the String equals() method. https://stackoverflow.com/questions/22030798/runtime-complexity-of-string-equals-in-java#:~:text=String.equals%20first%20compares%20the%20reference.%20If%20the%20reference,these%20there%20case%2C%20the%20complexity%20is%20O%20%281%29
equals(int num)	none	O(1)	

add(String num)	n=num.length()	O(n)	The method calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
subtract(String num)	n=num.length()	O(n)	The method calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
multiply(String num)	n=num.length()	O(n)	The method calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
divide(String num)	n=num.length()	O(n)	The method calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
calculate(String expression)	n=expression	O(n^3)	The method runs a for-loop dependent on the length of the input string. Within the loop, a class function is called that has a complexity of n and uses the String's substring as a parameter, so the overall complexity becomes n^3.
min(String value)	n=num.length()	O(n)	The method calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.
max(String num)	n=num.length()	O(n)	The method calls a class function that runs a for-loop which runs as many times as the length of the input string. The rest of the function is constant complexity, so the overall complexity is n.

```java
//Sohan Chatterjee
//SSW 315 Roman Numeral Class
//October 19, 2022
public class RomanNumber {
```

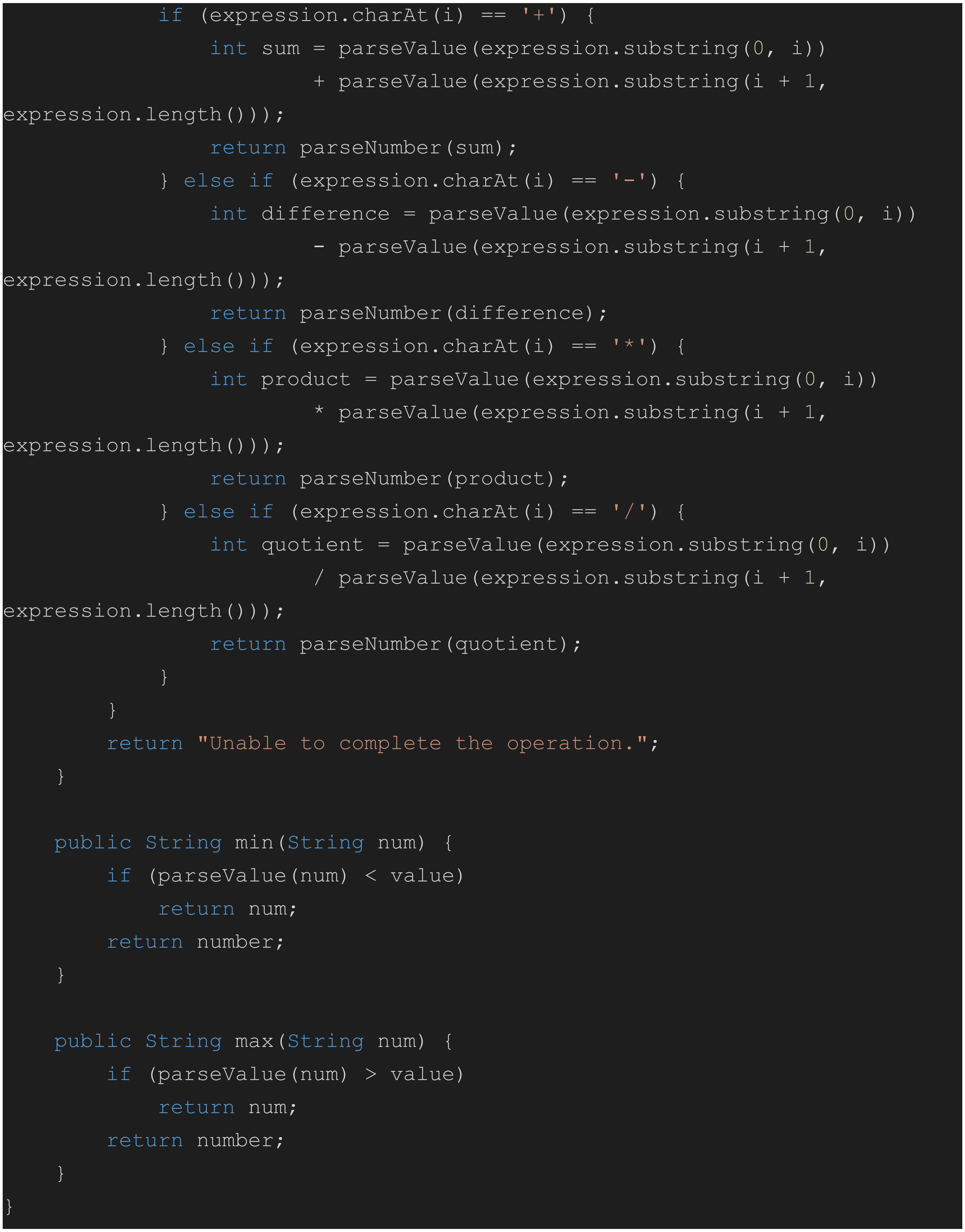


```
    private int value;
    private String number;

    public RomanNumber(int value) {
        this.value = value;
        this.number = parseNumber(value);
    }

    public RomanNumber(String number) {
        this.number = number;
        this.value = parseValue(number);
    }

    public int getValue() {
        return value;
    }

    public String getNumber() {
        return number;
    }

    public static int parseValue(String num) {
        int total = 0;
        for (int i = 0; i < num.length(); i++) {
            int first = getNum(num.charAt(i));
            if (i + 1 < num.length()) {
                int second = getNum(num.charAt(i + 1));
                if (first >= second)
                    total += first;
                else
                    total -= first;
            } else
                total += first;
        }
        return total;
    }

    public static int getNum(char r) {
        switch (r) {
            case 'I':
```



```
                return 1;
            case 'V':
                return 5;
            case 'X':
                return 10;
            case 'L':
                return 50;
            case 'C':
                return 100;
            case 'D':
                return 500;
            case 'M':
                return 1000;
            case 'ↁ':
                return 5000;
            case 'ↂ':
                return 10000;
            case ' ':
                return 50000;
            case ' ':
                return 100000;
        }
        return -1;
    }

    public static String parseNumber(int num) {
        if (num < 1 || num > 300000)
            return "Enter a valid number.";
        String[] hundredthousands = { "", " ", "  ", "   " };
        String[] tenthousands = { "", "ↂ", "ↂↂ", "ↂↂↂ", "ↂ ", " ",
" ↂ", " ↂↂ", " ↂↂↂ", "ↂ " };
        String[] thousands = { "", "M", "MM", "MMM", "Mↁ", "ↁ", "ↁM",
"ↁMM", "ↁMMM", "Mↂ" };
        String[] hundreds = { "", "C", "CC", "CCC", "CD", "D", "DC",
"DCC", "DCCC", "CM" };
        String[] tens = { "", "X", "XX", "XXX", "XL", "L", "LX", "LXX",
"LXXX", "XC" };
        String[] ones = { "", "I", "II", "III", "IV", "V", "VI", "VII",
"VIII", "IX" };
```

```
        return hundredthousands[num / 100000] + tenthousands[(num %
100000) / 10000] + thousands[(num % 10000) / 1000]
                + hundreds[(num % 1000) / 100] + tens[(num % 100) / 10] +
ones[num % 10];
    }

    public boolean equals(String num) {
        return (number.equals(num));
    }

    public boolean equals(int num) {
        return (value == num);
    }

    public String add(String num) {
        value += parseValue(num);
        number = parseNumber(value);
        return number;
    }

    public String subtract(String num) {
        value -= parseValue(num);
        number = parseNumber(value);
        return number;
    }

    public String multiply(String num) {
        value *= parseValue(num);
        number = parseNumber(value);
        return number;
    }

    public String divide(String num) {
        value /= parseValue(num);
        number = parseNumber(value);
        return number;
    }

    public static String calculate(String expression) {
        for (int i = 0; i < expression.length(); i++) {
```



```
            if (expression.charAt(i) == '+') {
                int sum = parseValue(expression.substring(0, i))
                        + parseValue(expression.substring(i + 1,
expression.length()));
                return parseNumber(sum);
            } else if (expression.charAt(i) == '-') {
                int difference = parseValue(expression.substring(0, i))
                        - parseValue(expression.substring(i + 1,
expression.length()));
                return parseNumber(difference);
            } else if (expression.charAt(i) == '*') {
                int product = parseValue(expression.substring(0, i))
                        * parseValue(expression.substring(i + 1,
expression.length()));
                return parseNumber(product);
            } else if (expression.charAt(i) == '/') {
                int quotient = parseValue(expression.substring(0, i))
                        / parseValue(expression.substring(i + 1,
expression.length()));
                return parseNumber(quotient);
            }
        }
        return "Unable to complete the operation.";
    }

    public String min(String num) {
        if (parseValue(num) < value)
            return num;
        return number;
    }

    public String max(String num) {
        if (parseValue(num) > value)
            return num;
        return number;
    }
}
```