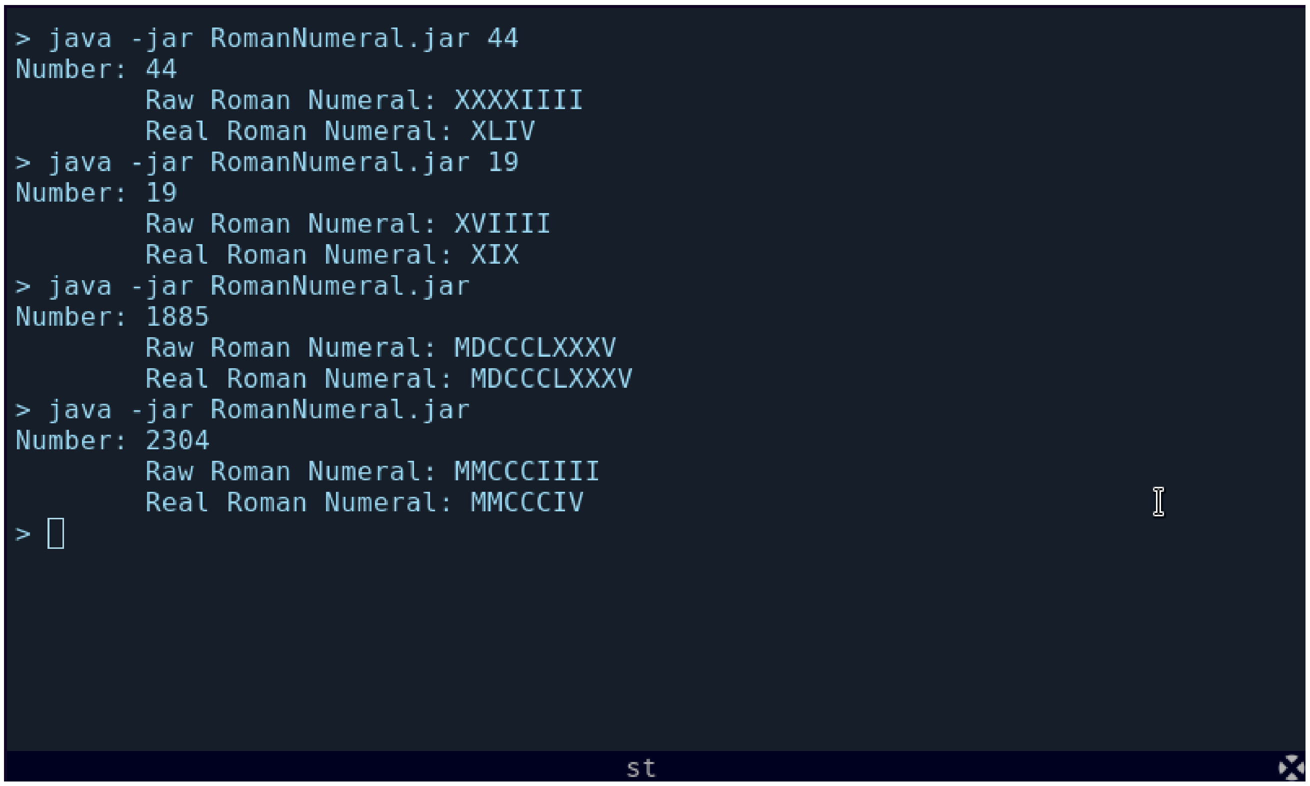
Project 6
Roman Numerals

For more information on the solution, please read **README.txt** *in the project submission*

Output



```
> java -jar RomanNumeral.jar 44
Number: 44
        Raw Roman Numeral: XXXXIIII
        Real Roman Numeral: XLIV
> java -jar RomanNumeral.jar 19
Number: 19
        Raw Roman Numeral: XVIIII
        Real Roman Numeral: XIX
> java -jar RomanNumeral.jar
Number: 1885
        Raw Roman Numeral: MDCCCLXXXV
        Real Roman Numeral: MDCCCLXXXV
> java -jar RomanNumeral.jar
Number: 2304
        Raw Roman Numeral: MMCCCIIII
        Real Roman Numeral: MMCCCIV
> 
```

RNTest.java

```
/**
* This program uses a replacement grammar technique to build
* valid roman numerals.
*
* @author Joe Smith and Kyle Guarco
*/
public class RNTest
{

    public static void main(String[] args)
    {
        int number = 1;

        if (args.length == 1)
        {
            try {
                // If there's an argument, use that as the number.
                number = Integer.parseInt(args[0]);

                if (number < 1 || number > 3999)
                    throw new NumberFormatException();
            } catch (NumberFormatException e) {
                System.out.println("Either that's not a number, or it isn't
between 1 and 3999.");
                System.exit(-1);
            }
        } else
            // If there's no arguments, choose a random number between 1 and
3999
            number = (int)(Math.random() * 4000d);

        IntegerToRomanNumeral numeral = new IntegerToRomanNumeral(number);

        // Print out the roman numeral
        String result = String.format("Number: %d\n\tRaw Roman Numeral: %s\n\
tReal Roman Numeral: %s",
number, numeral.getRawRomanNumeral(), numeral.getRomanNumeral());

        System.out.println(result);

        // To test out these solutions, look up "roman numeral converter"
        // To learn more about regex, look up "regex tester" and use the PHP
flavor
    }
}
```

IntegerToRomanNumeral.java

```java
import java.util.ArrayList;

/**
* This class converts integers to roman numerals using a simple replacement
* grammar
*
* @author Joe Smith and Kyle Guarco
*/
public class IntegerToRomanNumeral
{
    /** The values of roman numerals {M, D, C, L, X, V, I} */
    private static final int[] VALUES = {1000, 500, 100, 50, 10, 5, 1};
    /** All the possible roman numeral characters. */
    private static final char[] CHARS = {'M', 'D', 'C', 'L', 'X', 'V', 'I'};

    /** The original number */
    private int number;
    /** The numeral before any rules were applied */
    private String rawNumeral;
    /** The roman numeral string */
    private String numeralString;

    public IntegerToRomanNumeral(int number)
    {
        this.number = number;
        this.rawNumeral = createRawRomanNumeral();
        this.numeralString = applyGrammarRules(rawNumeral);
    }

    private String applyGrammarRules(String numeral)
    {
        // Grammar Rules:
        // AAAA → AB
        // BAB → AC

        // Apply grammar rules to replace invalid roman numeral sequences.

        // This should iterate the list of valid characters three times.
        for (int j = IntegerToRomanNumeral.CHARS.length - 1; j > 1; j -= 2)
        {
            char a = IntegerToRomanNumeral.CHARS[j];
            char b = IntegerToRomanNumeral.CHARS[j - 1];
            char c = IntegerToRomanNumeral.CHARS[j - 2];

            // These are regex rules that replace a sequence of characters.
            // This rule follows the 'AAAA' pattern above
            String abRule = String.format("%s{4}+", a);
            String abReplacement = "" + a + b;

            // This rule follows the 'BAB' pattern above
            String acRule = String.format("%s%s%s", b, a, b);
            String acReplacement = "" + a + c;
            // Apply the rules.
            numeral = numeral.replaceFirst(abRule, abReplacement);
            numeral = numeral.replaceFirst(acRule, acReplacement);
        }

        return numeral;
    }

    /** @return A raw roman numeral. */
    private String createRawRomanNumeral()
    {
        // Invoke fillNumeralCounter() so we know how much of
        // each character we need.
        int[] counter = fillNumeralCounter();

        // This will be the raw roman numeral.
        ArrayList<Character> romanNumeral = new ArrayList<>();

        // Begin writing the raw roman numeral representation.
        // counter.length == VALUES.length
        for (int i = 0; i < counter.length; i++)
        {
            char character = IntegerToRomanNumeral.CHARS[i];
            int charCount = counter[i];

            while (charCount > 0)
            {
                romanNumeral.add(character);
                charCount--;
            }
        }

        String rawNumeral = "";
        for (char c : romanNumeral)
            rawNumeral += c;

        return rawNumeral;
    }

    /** @return A counter for all possible numeral values (how many M's, D's,
etc...) */
    private int[] fillNumeralCounter()
    {
        // Create a counter that contains raw roman numerals.
        int[] counter = new int[IntegerToRomanNumeral.VALUES.length];
        // Copy the original number so we can use it without changing its value.
        int number = this.number;

        // Start subtracting the values in descending order.
        int counterIndex = 0;
        for (int val : IntegerToRomanNumeral.VALUES)
        {
            // Subtract the current value until you can't no more.
            while (number >= val)
            {
                number -= val;
                counter[counterIndex]++;
            }
            counterIndex++;
        }

        return counter;
    }

    public int getInteger()
    {
        return number;
    }

    public String getRawRomanNumeral()
    {
        return rawNumeral;
    }

    public String getRomanNumeral()
    {
        return numeralString;
    }
}
```