CSA 1017
Data Structures and Algorithms 1

Assignment

Mark Said Camilleri
B.Sc. (Hons) (Computing Science)
University of Malta

Friday 4th May 2016

Contents

Statement of Completion

The questions below were the ones that have been attempted:

Question 1	This question has been successfully completed.
Question 2	This question has been successfully completed.
Question 3	This question has been successfully completed.
Question 4	This question has been successfully completed.
Question 5	This question has been successfully completed.
Question 6	This question has been successfully completed.
Question 7	This question has been successfully completed.
Question 8	This question has been successfully completed.
Question 9	This question has been successfully completed.

Signature

Date

Task 1

Input	Expected Output	Actual Output
1	I	I
2	II	II
4	IV	IV
5	V	V
6	VI	IV
9	IX	IX
10	X	X
20	XX	XX
40	XL	XL
49	XLIX	XLIX
50	L	L
60	LX	LX
90	XC	XC
99	XCIX	XCIX
100	C	C
150	CL	CL
400	CD	CD
499	CDXCIX	CDXCIX
500	D	D
600	DC	DC
900	CM	CM
999	CMXCIX	CMXCIX
1000	M	M
1024	MXXIV	MXXIV
2000	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:
abc	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:	The input was not a valid number between 1 and 1024 Please try again and enter a number to convert:

The source code for Task 1

```java
/**
 * @author Mark Said Camilleri
 * @version 20160509
 */


import java.util.InputMismatchException;
import java.util.Scanner;


public class Question1 {

    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in);
        in.useDelimiter("\n");

        /*======================== WELCOME MESSAGE TO USER =======================*/
        System.out.println("-----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println("|---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri         |");
        System.out.println("|     Task 1: Arabic to Roman Numeral Converter     |");
        System.out.println("|---------------------------------------------------|");
        System.out.print("| Please enter a number between 1 and 1024: ");


        int toConvert = 0; //value to be converted.
        boolean isError; // temporary boolean value used for error checking of the input.
        do {
            isError = false;
            try {
                toConvert = in.nextInt();
            } catch (InputMismatchException e) {
                isError = true;
                in.next(); //To clear the buffer
            }
            /*=========== Makes sure input is a number is between 1 and 1024 =============*/
            if (isError || toConvert < 1 || toConvert > 1024) {
                /*=================== OUTPUT ERROR MESSAGE TO THE USER ==================*/
                System.out.println("|---------------------------------------------------|");
                System.out.println("|The input was not a valid number between 1 and 1024|");
                System.out.print("|Please try again and enter a number to convert: ");
            }
        } while (isError || toConvert < 1 || toConvert > 1024);

        System.out.printf("| %4d = %-24s in Roman Numerals |", toConvert, convert(toConvert));

    }

    /**
     * Takes an int decimal value and outputs a string of the same value in Roman Numerals.
     *
     * @param toConvert the decimal value to ve converted to Roman Numerals
     * @return The roman numeral equivalent of the input parameter
     */
    private static String convert(int toConvert) {

        //Defining the decimal and roman counterparts
        final int dec[] = {1, 4, 5, 9, 10, 40, 50, 90, 100, 400, 500, 900, 1000};
        final String rom[] = {"I", "IV", "V", "IX", "X", "XL", "L", "XC", "C", "CD", "D", "CM", "M"};

        /* Begins by checking the input paramerer against the largest roman numeral/numeral pair.
         * and works it's way down to the unit numeral.
         */
        for (int i = dec.length - 1; i >= 0; i--) {
            //If the value is larger, then the output is concatenated with the output of the difference.
            if (toConvert >= dec[i])
                return rom[i] + convert(toConvert - dec[i]);
        }
        return ""; //What to return at 0, the base case.
    }
}
```

Task 2

Input	Expected Output	Actual Output
4 5 +	9.0	9.0
12 6 -	6.0	6.0
3 2 /	1.5	1.5
7 2 *	14.0	14.0
23 85 + 92 *	9936.0	9936.0
43.5 3.2 - 4.5 * 3.24 +	184.59	184.59
34.8 62.11 * -76 /	-28.43984211	-28.439842105263157
282 -56 * 1024 - -55.6 *	934969.6	934969.6
Test	Your expression contained invalid characters. For input string "T" Your expression is invalid. Evaluation failed	Your expression contained invalid characters. For input string "T" Your expression is invalid. Evaluation failed
1 +	Stack is Empty. Your expression is invalid. Evaluation failed.	Stack is Empty. Your expression is invalid. Evaluation failed.
3 64 6 +	The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed.	The stack has not been emptied. There are too many operands in your expression. Your expression is invalid. Evaluation failed.
4 0 /	Infinity	Infinity

The source code for Task 2

Task 2.1 Stack Class

```java
import java.util.ArrayList;
import java.util.Collection;
import java.util.EmptyStackException;

/**
 * Created by mark on 14/02/16.
 *
 * A stack implemented as an ArrayList to have it dynamically increase
   its size.
 */
public class Stack<E> extends ArrayList {

    /**
     * Default constructor. Calls the ArrayList default constructor
     */
    public Stack() {
        super();
    }

    /**
     * Initialised a stack with the contents of the Collection in the
    parameter.
     *
     * @param c the contents to initialise the stack with.
     */
    public Stack(Collection<? extends E> c) {
        super(c);
    }

    /**
     * Pushes the data onto the stack
     *
     * @param data data to be pushed on the stack
     * @throws IndexOutOfBoundsException if not sucessfully pushed
     */
    public void push(E data) throws IndexOutOfBoundsException {
        int prevSize = this.size();
        this.add(data);

        //Condition to check if the data has been successfully added.
        if (!(this.size() == prevSize + 1))
            throw new IndexOutOfBoundsException("Failed to push to stack
    ");
    }

    /**
     * Pops the topmost item from the stack.
     * @return the data from the top of the stack is not empty.
     * @throws EmptyStackException if the stack is empty.
     */
    public E pop() throws ArrayIndexOutOfBoundsException {
        if (this.size() == 0) throw new EmptyStackException();
        else return (E) this.remove(this.size() - 1);
    }

    /**
     * Returns the data at the top of the stack without popping it.
     * @return the data if the stack is not empty. null if it is empty.
     */
    public E peek() {
        if (this.size() == 0) return null;
        else return (E) this.get(this.size() - 1);
    }

    /**
     * Returns a string representation of the contents of the stack.
     * @overrides toString() in class AbstractCollection<E>
     * @return The string representation of the ArrayList if not empty. "Stack is empty" if it is empty.
     */
    public String toString() {
        if (this.size() == 0) return "Stack is empty";
        else return super.toString();
    }
}
```

Task 2.2 Question 2 Main Class

```java
import java.util.EmptyStackException;
import java.util.Scanner;

/**
 * Created by mark on 09/02/16.
 * This answer assumes the RPN input is correct.
 */
public class Question2 {
    public static void main(String args[]) {

        //Initialize Scanner object
        Scanner in = new Scanner(System.in).useDelimiter("\n");

        /*======================= WELCOME MESSAGE TO USER =======================*/
        System.out.println(" -----------------------------------------------------");
        System.out.println("|    CSA 1017 - Data Structures and Algorithms 1    |");
        System.out.println(" |---------------------------------------------------|");
        System.out.println("|         Submission by Mark Said Camilleri          |");
        System.out.println("|     Task 2: Reverse Polish Notation evaluator      |");
        System.out.println(" |---------------------------------------------------|");
        System.out.println("|     Note: This program can only do +,-,* and /    |");
        System.out.print("| Please enter an expression to evaluate: "); //prompt for user input. Assumes correctness.

        /* Initialized a stack object (using the stack defined here).
```

```
         * Note, no importing of the Stack class.
         */
        Stack<Double> nums = new Stack<>();

        //Reads user input. Must be a valid RPN expression.
        String expression = in.next();
        System.out.println("
    |---------------------------------------------------|");
        System.out.println("| Contents of the stack at each step:
            |");//some message to user.

        boolean exceptionRaised = false; //used for error checking.
        try {
            //Iterates through the string inputted by the user.
            for (int i = 0; i < expression.length(); i++) {
                char cChar = expression.charAt(i);

                /* If the current character is a space,
                 * nothing needs to be done.
                 */
                if (Character.isWhitespace(cChar)) continue;

                /* If it's a '+', then 2 numbers are popped, added and
                 * the answer is pushed onto the stack.
                 */
                else if (cChar == '+') {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 + num1);
                }
                /* If it's a '-', then 2 numbers are popped,
                 * subtracted and the answer is pushed onto the stack.
                 * The conjunction is to make sure that it's not
                 * detecting a negative number. The disjunction is
                 * true if the '-' is at the end of the string or
                 * there is a space after it. These both make sure
                 * that the '-' does not belong to a negative number
                 */
                else if (cChar == '-' && (i == expression.length() - 1
    || Character.isWhitespace(expression.charAt(i + 1)))) {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 - num1);
                }
                /* If it's a '*', then 2 numbers are popped,
                 * multiplied and the answer is pushed onto the stack.
                 */
                else if (cChar == '*') {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 * num1);
                }
                /* If it's a '/', then 2 numbers are popped,
                 * divided and the answer is pushed onto the stack.
                 */
```

```java
                else if (cChar == '/') {
                    double num1 = nums.pop();
                    double num2 = nums.pop();

                    nums.push(num2 / num1);
                }
            /*
             * Otherwise, assuming it's inputted correctly, the
             * character must be a number. In which case it is
             * converted ot a double (allowing for any real number
             * to be inputted) and pushed onto the stack.
             */
                else {
                    int start = i++;
                    while (Character.isDigit(expression.charAt(i)) || expression.charAt(i) == '.')
                        i++;

                    nums.push(Double.parseDouble(expression.substring(start, i)));
                }
                System.out.printf("| %-49s |\n", nums.toString());
            }
            if (nums.size() != 1) {
                System.out.println("| The stack has not been emptied. There are too      |");
                System.out.println("| many operands in your expression.                 |");
                exceptionRaised = true;
            }

        } catch (IndexOutOfBoundsException ioobe) {
            System.out.printf("| %-49s |\n", ioobe.getMessage());
            exceptionRaised = true;
        } catch (EmptyStackException ese) {
            System.out.println("| Stack is Empty.                                   |");
            exceptionRaised = true;
        } catch (NumberFormatException nfe) {
            System.out.println("| Your expression contained invalid characters.     |");
            System.out.printf("| %-49s |\n", nfe.getMessage());
            exceptionRaised = true;
        } finally {
            if (exceptionRaised) {
                System.out.println("| Your expression is invalid. Evaluation failed.    |");
                System.out.println("

");
                System.exit(1);
            }
        }
        /*When the above iteration is complete, there should only be
         * one item on the stack whcih is the answer.
         */
        System.out.println("|---------------------------------------------------|");
        System.out.printf("| Answer of Evaluation = %-26s |\n", nums.pop().toString());
        System.out.println("|---------------------------------------------------|");
    }
}
```

Task 3

Input	Expected Output	Actual Output