# Object Oriented Programming
# Assignment 9 - RPN Calculator - Model

Mr. D Bylsma
Chris
Adrian

## 1 Hmmmmmm

We are used to infix notation - "3 + 4" - where the operator is between the operands. There is also prefix notation, where the operand comes before the operands. A Reverse Polish Notation calculator uses postfix notation, and was one of the first handheld calculators built. At first, it seems confusing, but is very logical once you think about it. Instead of doing "2 + 3 + 4", you may do "2 [enter] 3 [enter] 4 [enter] + +". You will be implementing an RPN calculator for this assignment.

## 2 How does an RPN calculator work?

It is quite simple in principle. The calculator keeps a stack - a list of numbers. When you type a number and hit "enter", it *pushes*, or appends the number to the stack. So you build up a *stack* of numbers. Whenever you click an operand, it applies the operator to the top of the stack. In the previous example, it builds a stack like [2, 3, 4]. When you hit the first "+", it *pops* off the top/most recent two elements off the list and "pluses" them. Lastly, it *pushes* the result back on the stack, so it looks like [2, 7]. When you hit plus again, it pops off the two elements (so the stack is temporarily empty), adds them, and pushes it back on the stack, so you get [9].

## 3 What you need to do

For the first part of this assignment, think about what classes you need. Java has a Stack class for you, but write your own. Use encapsulation, think about what methods it should have, and call it something like CalculatorStack. Add an option to "roll" the stack; shift it left or right by one (and the end number *rollls*) to the other end). Other classes may include Controller, Handler, or SpecialOperationsHandler.

## 4 Why?

- It's one of the better assignments I can think of as a teaching tool
- RPN calculators are awesome - they're far more logical than infix calculators once you get used to them
- You should learn RPN calculators; besides the historical importance, it forces you to think differently

## 5 Tips

Here are some interfaces you should implement (the way to break down the tasks into classes and methods)

```
/**
 * - A stack is what it literally sounds like - for example, a stack of dishes.
 * When you add something, it goes to the top
 * When you want to remove something, you only take what's at the top of the stack
 * - I recommend using a List<Double> to save the numbers (for example: an ArrayList<Double>)
 */
public interface RPNStack {

        /**
         * Adds a number to the top of the stack
         */
```

```
        public void push(double num);
        /**
         * Returns the number at the top of the stack, and removes it
        public double pop();
        /**
         * Returns the number at the top of the stack
         */
        public double peek();
        /**
         * Removes the number at the top of the stack
         */
        public void drop();
        /**
         * Switches the top two items
         */
        public void swap();
        /**
         * The top element of the stack gets inserted at the bottom,
         * and the previously-second item is now at the top
         */
        public void rollUp();
        /**
         * The bottom (last) element of the stack is now at the top,
         * and everything else is shifted down one
         * (the previously-second-last item is now at the bottom)
         */
        public void rollDown();
        /**
         * Returns an array of 4 numbers that may be used as a preview for the stack
         */
        public double[] getPreview();

        /**
         * -Returns a formatted String with all data about the state of this object
         * for debugging
         */
        public String getDebug();
}

/**
 * The RPNCalculator class is independent to the application;
 * It will include all the functionality regarding a calculator,
 * but nothing about how it's displayed or activated
 * I've written some non-interface code here as scaffolding
public interface RPNCalculator {
        private RPNStack stack;

        /**
         * These methods don't need number parameters because you get them from the stack
         */
        public void add() {
                public double first = this.stack.pop(); // returns first number, and also remove
                public double second = this.stack.pop(); // returns what was the second number,
                this.stack.push(first + second); // adds it back on the stack
        }
        public void subtract();
        public void divide();
        public void multiply();

        /**
```

```
         * This should also contain all the public methods of the RPNStack class;
         * it seems like repeating itself, but just do it
         */
        public void push(double num) {
                this.stack.push(num);
        }
        public double pop() {
                return this.stack.pop();
        }
        ...

        public String getDebug();
}

public interface RPNController {
        private RPNCalculator calc;
        /**
         * - This gets whatever the user typed in
         * - For simplicity, you may want to make all you commands single letters:
         * if it is a number, process it as is, but if it is a String, loop over
         * it and and call process on each of the characters
         * - I have written partial code for this, although in a real interface there
         * would not be code
         */
        public void processInput(String input) {
                try {
                        double num = Double.parseDouble(input);
                        // We have a valid number
                        // [code for processing entering a new number, you probably want]
                        this.calc.push(num);
                        return;
                } catch (NumberFormatException ex) {
                        // We didn't talk about numbers, but basically this "catch" block
                        // will execute if the String was not a number
                }
                for (int i = 0; i < input.length(); i++) {
                        this.processInput(input.charAt(i));
                }
        }

        /**
         * - This is an overloaded method, broken down for decomposition
         * - Java knows it is different because of the "signature";
         * it has the same name but different parameters
         */
        private void processInput(char input) {
                if (input == 'a') {
                        this.calc.add();
                } else if (input == 'h') {
                        this.showHelp();
                } ... {
                        ...
                } else {
                        // invalid command (ignore it)
                }
        }

        /**
         * - Returns a String: a text-display of the calculator, such as a preview of the stack
         * - This may be used in the console (via a command) or via some external class
```

```
         * - Hint: use "\n" for new lines
         */
        public String getDisplay();
        /**
         * - Gets help instructions
         */
        public String getHelp();
        /**
         * This should include the info of this.calc.getDebug();
         */
        public String getDebug();
}

public class RPNDriver {
        public static void main(String[] args) {
                RPNController controller = new RPNController();
                SimpleConsole.println("Welcome");
                while (true) {
                        String input = SimpleConsole.readLine("");
                        // Notice that the controller of the calculator is not responsible for s
                        if (input.equals("quit")) {
                                break;
                        }
                        controller.processInput(input);
                }
                SimpleConsole.println("Done. This is your stack");
        }
}
```