Patel200921323Short8

December 8, 2020

1 Short Assessed Exercise

2 Level 8

2.1 Jatinkumar Patel

2.2 06/12/2020

2.3 Version 1

2.4 Summary of the Question

Using recursion, create a calculator that takes in Reverse Polish Notation expressions and returns the result of the expression to the user. The calculator should follow this recursively defined language:

EXP = + DIGIT EXP | - DIGIT EXP | & EXP | DIGIT

DIGIT = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

2.5 The literate program development

2.5.1 inputString

What it does *A string input method that we can call from anywhere in the program with the given string parameter message. It will prompt the user for input and return the user's answer to whichever method has asked for it.*

Implementation (how it works) *Takes a String argument as a parameter. This string will be used as the prompt for the user to enter input. A new scanner is declared and initialised, after which the prompt is outputted to the user. The user's input is not saved inside the method and instead is directly returned to the method that called inputString.*

[2]:
```
// String input method to have cleaner code
public static String inputString (String message) {
    Scanner scanner = new Scanner(System.in);
    System.out.println(message);
    return scanner.nextLine();
}
```

Testing

[13]:
```
inputString("Enter some input: ");
```

```
Enter some input:
some
```

[13]:
```
some
```

2.5.2 evalExp

What it does *Evaluates the expression that has been inputted. It checks the length of the input, and calls either evalDigit or evalOperator based on this.*

Implementation (how it works) *We need an integer result variable and another integer variable that stores the length of the input for comparison. If the length is equal to 1, evalDigit is called and stored in result. Otherwise, evalOperator is called and stored in result.*

[3]:
```
// Checks if the expression is a digit or if calculations are needed
public static int evalExp(String input) {
    int result;
    int length = input.length();

    if (length == 1) {
        result = evalDigit(input);
    } else {
        result = evalOperator(input);
    }
    return result;
}
```

Testing

[14]:
```
evalExp("+23");
```

[14]:
```
5
```

2.5.3 evalOperator

What it does *Checks the operator (the first character of the input) to see which evaluation needs to be performed and returns the result of that evaluation.*

Implementation (how it works) *An integer result is initialised at a default of -1. The first character of the string input, which was taken as a parameter, is checked to see if it is either a +, - or &, which are the only valid operators for this recursively defined language. If the character is not equal to any of these, the program quits. Otherwise, it will call the respective method to add, subtract or sum the expression.*

[4]:
```
// Checks the operator to see which evaluation needs to be done
public static int evalOperator (String input) {
```

```
    int result = -1;
    if (input.charAt(0) == '+') {
        result = evalAdd(input);
    } else if (input.charAt(0) == '-') {
        result = evalSub(input);
    } else if (input.charAt(0) == '&') {
        result = evalSum(input);
    } else { // Quits if operator is invalid
        quit();
    }
    return result;
}
```

Testing

[15]:
```
evalOperator("+23");
```

[15]: 5

2.5.4 evalDigit

What it does *Takes a string input and converts it to it's integer form if possible. Returns the integer variable.*

Implementation (how it works) *An integer variable is declared and initialised with a default value of -1. We then check the input to see if it is equal to a number from 0 - 9 (inclusive). If it is equal to any of the digits, the value of result is changed to that digit. Otherwise, there is a problem with the expression and so the program will call the quit method.*

[5]:
```
// Returns the digit in integer form
public static int evalDigit (String input) {
    int result = -1;

    if (input.equals("0")) {
        result = 0;
    } else if (input.equals("1")) {
        result = 1;
    } else if (input.equals("2")) {
        result = 2;
    } else if (input.equals("3")) {
        result = 3;
    } else if (input.equals("4")) {
        result = 4;
    } else if (input.equals("5")) {
        result = 5;
    } else if (input.equals("6")) {
        result = 6;
```

```
    } else if (input.equals("7")) {
        result = 7;
    } else if (input.equals("8")) {
        result = 8;
    } else if (input.equals("9")) {
        result = 9;
    } else {
        quit();
    }
    return result;
}
```

Testing

[16]:
```
evalDigit("3");
```

[16]:
```
3
```

2.5.5 evalAdd

What it does *Performs an addition evaluation on the input that is given as a parameter. The addition rule states that DIGIT can be followed by EXP, so we need to make sure that we check if this occurs when performing the evaluation.*

Implementation (how it works) *Three integers are declared, one to store the result, one to store the first number to add, and one to store the second number to add. We then split up the input into different strings, depending on the character. Since the operator is always first in the expression, we can split it off using nextChar and storing it in a variable (operator) of its own. The rest of the string can be put into another variable (next) using restChars. We then know that + is always followed by a digit, so we split the digit off from the rest of the remaining string just like we did with the operator. The digit is stored in string first and the rest of the string is stored in another string end.*

We now assign the first digit to num1 using evalDigit, and check the operator once more to make sure that it is the correct operator for this evaluation. Since the string end can have another expression stored in it, we assign the second digit using evalExp, which will get the result of the expression and store it in num2. After that, we just add num1 and num2, store the value in result and return the variable result.

[6]:
```
// Performs the addition evaluation
public static int evalAdd (String input) {
    int result;
    int num1;
    int num2;

    String operator = nextChar(input); // Operator is always first
    String next = restChars(input);
    String first = nextChar(next); // First integer
```

```
    String end = restChars(next); // Expression

    num1 = evalDigit(first);
    checkOperator(operator);
    num2 = evalExp(end);

    result = num1 + num2;
    return result;
}
```

Testing

[17]:
```
evalAdd("+23");
```

[17]: 5

2.5.6 evalSub

What it does *Performs a subtraction evaluation on the input that is given as a parameter. The addition rule states that DIGIT can be followed by EXP, so we need to make sure that we check if this occurs when performing the evaluation.*

Implementation (how it works) *Three integers are declared, one to store the result, one to store the first number to subtract from, and one to store the second number to subtract. We then split up the input into different strings, depending on the character. Since the operator is always first in the expression, we can split it off using nextChar and storing it in a variable (operator) of its own. The rest of the string can be put into another variable (next) using restChars. We then know that - is always followed by a digit, so we split the digit off from the rest of the remaining string just like we did with the operator. The digit is stored in string first and the rest of the string is stored in another string end.*

We now assign the first digit to num1 using evalDigit, and check the operator once more to make sure that it is the correct operator for this evaluation. Since the string end can have another expression stored in it, we assign the second digit using evalExp, which will get the result of the expression and store it in num2. After that, we just subtract num2 from num1, store the value in result and return the variable result.

[7]:
```
// Completes a subtraction evaluation
public static int evalSub (String input) {
    int result;
    int num1;
    int num2;

    String operator = nextChar(input); // Operator is always first
    String next = restChars(input);
    String first = nextChar(next); // First integer
    String end = restChars(next);
```

```
    num1 = evalDigit(first);
    checkOperator(operator);
    num2 = evalExp(end);

    result = num1 - num2;
    return result;
}
```

Testing

[18]:
```
evalSub("-85");
```

[18]: 3

2.5.7 evalSum

What it does *Returns the sum to n of the given digit. If there are expressions to do before summing to n, then those will be checked and performed.*

Implementation (how it works) *We need two integer variables, one to store the result and one to store the number that we are summing up to. We split the operator off the input and then use evalExp on the rest of the string to make sure that all expressions are evaluated before we try to calculate the sum. We then check the operator to make sure that it is the correct one for this evaluation. We then call the method sumOfN to calculate the sum up numbers up to the given input numToSum. This is stored in result and returned back to the method which called evalSum.*

[8]:
```
// Evaluating anything after &
public static int evalSum (String input) {
    int result;
    int numToSum;

    String operator = nextChar(input);
    String next = restChars(input);
    numToSum = evalExp(next);

    checkOperator(operator);
    result = sumOfN(numToSum);
    return result;
}
```

Testing

[19]:
```
evalSum("&7");
```

[19]: 28

2.5.8 sumOfN

What it does *Calculates the sum of numbers up to a certain number, which is given in the input. Does this by recursively calling itself until the number to add to the result goes down to 0.*

Implementation (how it works) *Takes an integer number as a parameter. As long as the number is not equal to 0, it will keep adding to the result and calling itself again. When the number to add reaches 0, the recursive calls stop and the result of adding is returned.*

[9]:
```
// Sum of numbers up to n
public static int sumOfN (int number) {
    if (number != 0) {
        return number + sumOfN(number - 1);
    } else {
        return number;
    }
}
```

Testing

[20]:
```
sumOfN(7);
```

[20]: 28

2.5.9 restChars

What it does *Returns all but the first character of a string.*

Implementation (how it works) *Takes the original string as a parameter, and using the built-in method substring (which takes the starting character's index as a parameter), returns all characters from index 1 to the end of the string.*

[10]:
```
// Returns all but the first character of the string input.
public static String restChars(String input) {
    return input.substring(1);
}
```

Testing

[21]:
```
restChars("input");
```

[21]: nput

2.5.10 nextChar

What it does *Returns the first character of a given string.*

Implementation (how it works) *Takes the original string as a parameter, then, using the built-in method substring (which can also take two indexes as parameters), returns the character(s) from index 0 to index 1. In this case that will just return the first character of any given string.*

[11]:
```
// Returns the first character of the string input.
public static String nextChar (String input) {
    return input.substring(0,1);
}
```

Testing

[22]:
```
nextChar("input");
```

[22]:
```
i
```

2.5.11 checkOperator

What it does *Checks the operator to make sure it is either a +, - or &, which are the only valid operators in our recursively defined language.*

Implementation (how it works) *Takes a string input as a parameter. Checks the first character of this string by using built-in method charAt(0), and if the character is not a +, - or &, then there is something wrong with the expression and the quit method will be called.*

[12]:
```
// Making sure the operator is valid.
public static void checkOperator (String input) {
    if (!(input.charAt(0) == '+' || input.charAt(0) == '-' || input.charAt(0)␣
 ↪== '&')) {
        quit();
    }
}
```

Testing

[23]:
```
checkOperator("&");
```

2.5.12 quit

What it does *Outputs an error message and ends the program safely.*

Implementation (how it works) *Outputs a simple statement to tell the user that the expression they entered is invalid, then exits the program to make sure nothing keeps running.*

[13]:
```
// Quits the program and shuts down processes safely.
public static void quit(){
    System.out.println("There is something wrong with the expression you␣
 ↪entered.");
    System.exit(0);
```

```
}
```

Testing

[ ]:
```
quit();
```

```
There is something wrong with the expression you entered.
```

2.5.13 Running the program

Run the following call to simulate running the complete program.

[14]:
```
String input = inputString("Please input the expression: ");
int answer = evalExp(input);

System.out.println("The answer is " + answer);
```

```
Please input the expression:
+23
The answer is 5
```

2.6 The complete program

This version will only compile here. To run it copy it into a file called initials.java on your local computer and compile and run it there.

[ ]:
```
/*
Jatinkumar Patel
03/12/2020
Version 1
Recursive parser program that takes Reverse Polish Notation expressions as input
and returns the result of evaluating it to the user.
*/

import java.util.Scanner; // Needed to make scanner available

public class RPNEvaluator {
    public static void main(String[] args) {
        String input = inputString("Please input the expression: ");
        int answer = evalExp(input);

        System.out.println("The answer is " + answer);
        System.exit(0);
    }

    // String input method to have cleaner code
    public static String inputString (String message) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.println(message);

        return scanner.nextLine();
    }

    // Checks if the expression is a digit or if calculations are needed
    public static int evalExp(String input) {
        int result;
        int length = input.length();

        if (length == 1) {
            result = evalDigit(input);
        } else {
            result = evalOperator(input);
        }
        return result;
    }

    // Checks the operator to see which evaluation needs to be done
    public static int evalOperator (String input) {
        int result = -1;
        if (input.charAt(0) == '+') {
            result = evalAdd(input);
        } else if (input.charAt(0) == '-') {
            result = evalSub(input);
        } else if (input.charAt(0) == '&') {
            result = evalSum(input);
        } else { // Quits if operator is invalid
            quit();
        }
        return result;
    }

    // Returns the digit in integer form
    public static int evalDigit (String input) {
        int result = -1;

        if (input.equals("0")) {
            result = 0;
        } else if (input.equals("1")) {
            result = 1;
        } else if (input.equals("2")) {
            result = 2;
        } else if (input.equals("3")) {
            result = 3;
        } else if (input.equals("4")) {
            result = 4;
```

```
        } else if (input.equals("5")) {
            result = 5;
        } else if (input.equals("6")) {
            result = 6;
        } else if (input.equals("7")) {
            result = 7;
        } else if (input.equals("8")) {
            result = 8;
        } else if (input.equals("9")) {
            result = 9;
        } else {
            quit();
        }
        return result;
    }

    // Performs the addition evaluation
    public static int evalAdd (String input) {
        int result;
        int num1;
        int num2;

        String operator = nextChar(input); // Operator is always first
        String next = restChars(input);
        String first = nextChar(next); // First integer
        String end = restChars(next); // Expression

        num1 = evalDigit(first);
        checkOperator(operator);
        num2 = evalExp(end);

        result = num1 + num2;
        return result;
    }

    // Completes a subtraction evaluation
    public static int evalSub (String input) {
        int result;
        int num1;
        int num2;

        String operator = nextChar(input); // Operator is always first
        String next = restChars(input);
        String first = nextChar(next); // First integer
        String end = restChars(next);

        num1 = evalDigit(first);
```

```
        checkOperator(operator);
        num2 = evalExp(end);

        result = num1 - num2;
        return result;
    }

    // Evaluating anything after &
    public static int evalSum (String input) {
        int result;
        int numToSum;

        String operator = nextChar(input);
        String next = restChars(input);
        numToSum = evalExp(next);

        checkOperator(operator);
        result = sumOfN(numToSum);
        return result;
    }

    // Sum of numbers up to n
    public static int sumOfN (int number) {
        if (number != 0) {
            return number + sumOfN(number - 1);
        } else {
            return number;
        }
    }

    // Returns all but the first character of the string input.
    public static String restChars(String input) {
        String rest = input.substring(1);
        if (rest.equals("")){
            return "Empty";
        }
        return rest;
    }

    // Returns the first character of the string input.
    public static String nextChar (String input) {
        return input.substring(0,1);
    }

    // Making sure the operator is valid.
    public static void checkOperator (String input) {
        if (!(input.charAt(0) == '+' || input.charAt(0) == '-' ||
```

```
                input.charAt(0) == '&')) {
            quit();
        }
    }

    // Quits the program and shuts down processes safely.
    public static void quit(){
        System.out.println("There is something wrong with the expression you␣
↪entered.");
        System.exit(0);
    }
}
```

END OF LITERATE DOCUMENT