Lab9

April 4, 2024

1 Lab 9

Deadline: **Week 11** in your respective lab session

1.0.1 Name:Hansel Rodrigues

1.0.2 Student ID:230603866

Note that this lab contains only 3 questions, but the total number of marks is still 5. The amount of marks does not correspond to the difficulty of the question but rather its importance. The higher the mark, the higher the importance.

1.1 Question 1 [2 mark]

You are given a partially implemented class `Level`. Implement the class method `loadLevel` and instance method `saveLevel`.

`loadLevel` should attempt to read from `levelSave.txt` and create an instance of `Level` with the value read from the file. If an exception occurs during the reading process, `loadLevel` should create an instance of `Level` with a default value `0`.

`saveLevel` should write the value stored inside the `level` attribute into `levelSave.txt`.

Note that class `Level` has two constructors; you are expected to use them as appropriate.

You are expected to understand how the `Level` class fits into the overall program and its expected behaviour.

Write your answer below:

[8]:
```
import java.io.*;

class Level {
    private int level;

    private Level(int level) {
        this.level = level;
    }
```

```
    public Level() {
        this.level = 0;
    }

    public void nextLevel() {
        this.level++;
    }

    public int getLevel() {
        return this.level;
    }

    public static Level loadLevel()
    {
        try
        {
            BufferedReader reader = new BufferedReader(new␣
↪FileReader("levelSave.txt"));
            int loadedLevel = Integer.parseInt(reader.readLine());
            reader.close();
            return new Level(loadedLevel);
        }
        catch (IOException | NumberFormatException e)
        {
            return new Level();
        }
    }

    public void saveLevel()
    {
        try
        {
            BufferedWriter writer = new BufferedWriter(new␣
↪FileWriter("levelSave.txt"));
            writer.write(Integer.toString(this.level));
            writer.close();
        }
        catch (IOException e)
        {
            System.err.println("Error saving level: " + e.getMessage());
        }

    }

}
```

Run your program:

[9]:
```
public class Main1 {
    public static void main(String[] args) {
        String option = inputString("1) New Game\n2) Load Game");

        Level level = option.equals("1") ? new Level() : loadGame();

        String answer;
        do {
            answer = inputString("Do you want to progress to the next level? y/
 ↪n?");

            if (answer.equals("y")) {
                level.nextLevel();
                System.out.println("You are on level " + level.getLevel());
            }
            else {
                level.saveLevel();
            }
        } while (answer.equals("y"));
    }

    public static Level loadGame() {
        return Level.loadLevel();
    }

    public static String inputString(String message) {
        System.out.println(message);
        return new Scanner(System.in).nextLine();
    }
}
```

[10]:
```
Main1.main(null);
```

```
1) New Game
2) Load Game
 1
Do you want to progress to the next level? y/n?
 y
You are on level 1
Do you want to progress to the next level? y/n?
 y
You are on level 2
Do you want to progress to the next level? y/n?
 n
```

1.2 Question 2 [2 mark]

Write a program that recursively parses expressions, input as strings, from the following recursively defined language and calculates and prints out the answer to the calculations.

```
<EXP> = + <DIGIT> <EXP> | - <DIGIT> <EXP> | * <DIGIT> <EXP> | / <DIGIT> <EXP> | ^ <DIGIT> <EXP
<DIGIT> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F
```

Here EXP represents expressions, where: - `+` represents addition. - `-` represents subtraction. - `&` represents sum up to the given number (inclusive). - `*` represents multiplication. - `/` represents division. - `^` represents exponentiation. - `%` represents modulus (remainder after division).

DIGIT represents a number from 0 to 15.

Legal expressions in this language involve putting the operator before its arguments (this is called Polish notation). Instead of writing 1+2, in this language you write +12. Notice only single digits are allowed, and spaces are not allowed.

An example run of the program (characters in bold are typed in by the user):
`Please input the expression` **`3`**
`The answer is 3`

Another example run:
`Please input the expression` **`B`**
`The answer is 11`

Another example run:
`Please input the expression` **`&3`**
`The answer is 6`

Explanation: The & operator calculates the sum up to the given number (inclusive), so &3 sums up $1 + 2 + 3 = 6$.

Another example run:
`Please input the expression` **`&+23`**
`The answer is 15`

Explanation: Here, +23 represents addition (+) of 2 and 3, resulting in 5. Then, &5 sums up $1 + 2 + 3 + 4 + 5 = 15$.

Another example run:
`Please input the expression` **`^2+D9`**
`The answer is 4194304`

Explanation: Here, ^2+D9 calculates 2 raised to the power of the result of +D9. +D9 represents the addition of 13 (D in hexadecimal) and 9, resulting in 22. So, the expression calculates $2^{22} = 4194304$.

Another example run:
`Please input the expression` **`&+1-82`**
`The answer is 28`

Explanation: The expression &+1-82 is parsed as (& (+ 1 (- 8 2))). First, the expression inside the innermost parentheses is evaluated: (- 8 2) calculates the subtraction of 8 - 2, resulting in 6. Next, the expression inside the next parentheses is evaluated: (+ 1 6) calculates the addition of 1 + 6, resulting in 7. Then, the expression inside the outermost parentheses is evaluated: &7 calculates the sum up to the given number (inclusive), so &7 sums up 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28.

Another example run:

```
Please input the expression *2&%85
The answer is 12
```

Explanation: Here, *2&%85 first calculates the result of %85, which represents the modulus of 8 by 5, resulting in 3. Then, &3 calculates the sum up to 3, which is 1 + 2 + 3 = 6. Finally,* 26 calculates the multiplication of 2 and 6, resulting in 12.

Another example run:

```
Please input the expression +C*7/A%92
The answer is 82
```

Explanation: This expression involves addition, multiplication, division, and modulus. The program evaluates each operation step by step and outputs the final result as 82.

Use a regular expression to check if the inputted string contains only valid characters. Additionally, if all characters are valid but the expression is illegal, throw a custom exception (you need to define it); you should not catch it.

You must not use an explicit loop at all in your program.

Hint: What base do all the digits belong to? How can you easily parse the string without writing much code?

Write your answer below:

[141]:

Run your program:

[ ]:

1.3 Question 3 [1 mark]

Consider the triangle below, where the maximum total from top to bottom is demonstrated:

3 7 4 2 4 6 8 5 9 3

The maximum total, obtained by summing the highlighted numbers (3 + 7 + 4 + 9 = 23), is 23.

Now, your task is to find the maximum total from top to bottom of a larger triangle, represented below:

75 95 64 17 47 82 18 35 87 10 20 04 82 47 65 19 01 23 75 03 34 88 02 77 73 07 63 67 99 65 04 28
06 16 70 92 41 41 26 56 83 40 80 70 33 41 48 72 33 47 32 37 16 94 29 53 71 44 65 25 43 91 52 97
51 14 70 11 33 28 77 73 17 78 39 68 17 57 91 71 52 38 17 14 91 43 58 50 27 29 48 63 66 04 68 89
53 67 30 73 16 69 87 40 31 04 62 98 27 23 09 70 98 73 93 38 53 60 04 23

Start by running the `save` method below, which will save the above triangle into a text file called `triangle.txt`. Allow it a few seconds to appear in your current directory.

Then, your objective is to implement the `load` method, which should load the data from the `"triangle.txt"` file and return a 2D integer array representing the triangle above.

Using the 2D array, solve the problem stated above (the maximum total from top to bottom of the triangle).

Hint: Start from the bottom of the triangle.

Write your answer below:

[4]:
```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
public static void save() {
        String data = "75\n" +
                      "95 64\n" +
                      "17 47 82\n" +
                      "18 35 87 10\n" +
                      "20 04 82 47 65\n" +
                      "19 01 23 75 03 34\n" +
                      "88 02 77 73 07 63 67\n" +
                      "99 65 04 28 06 16 70 92\n" +
                      "41 41 26 56 83 40 80 70 33\n" +
                      "41 48 72 33 47 32 37 16 94 29\n" +
                      "53 71 44 65 25 43 91 52 97 51 14\n" +
                      "70 11 33 28 77 73 17 78 39 68 17 57\n" +
                      "91 71 52 38 17 14 91 43 58 50 27 29 48\n" +
                      "63 66 04 68 89 53 67 30 73 16 69 87 40 31\n" +
                      "04 62 98 27 23 09 70 98 73 93 38 53 60 04 23";
    try {
        FileWriter writer = new FileWriter("triangle.txt");
        writer.write(data);
        writer.close();
    } catch (IOException e) {}
}

save();
```

[5]:
```
public static int[][] load() throws IOException {
    Scanner scanner = new Scanner(new FileReader("triangle.txt"));
    int rows = 0;
    while (scanner.hasNextLine()) {
      rows++;
      scanner.nextLine();
    }
```

```
    int[][] triangleArray = new int[rows][];
    scanner = new Scanner(new FileReader("triangle.txt"));

    for (int i = 0; i < rows; i++) {
      String line = scanner.nextLine();
      String[] numbers = line.split(" ");
      triangleArray[i] = new int[numbers.length];
      for (int j = 0; j < numbers.length; j++) {
        triangleArray[i][j] = Integer.parseInt(numbers[j]);
      }
    }
    scanner.close();

    return triangleArray;
  }
```

[6]:
```
public static void solvePuzzle(int[][] triangleArray)
{
    for (int i = triangleArray.length - 2; i >= 0; i--) {
      for (int j = 0; j < triangleArray[i].length; j++) {
        // Add the maximum value from the children nodes
        triangleArray[i][j] += Math.max(triangleArray[i + 1][j],␣
 ↪triangleArray[i + 1][j + 1]);
      }
    }

    System.out.println("Maximum path sum: " + triangleArray[0][0]);
}
```

Run your program:

[11]:
```
solvePuzzle(load());
```

```
Maximum path sum: 1074
```

[ ]:
```

```