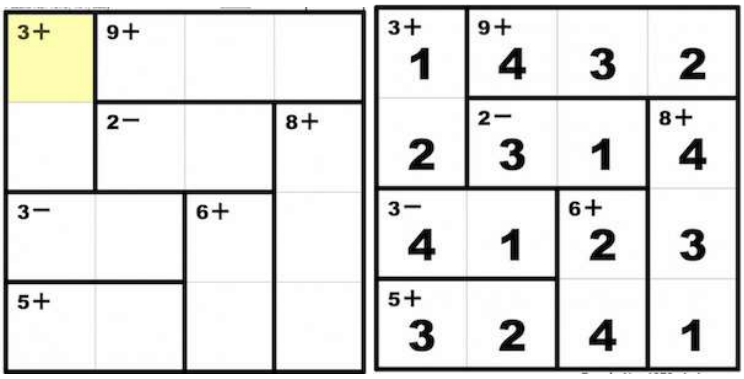
CSCI-140 Computer Science AP
CSCI-242 Computer Science Transfers
Lab 4 - KenKen



Overview

In this lab you will write a complete implementation for a KenKen puzzle verifier. Given a puzzle with a potential solution, your program will verify the correctness of each region

The official site contains the rules for how to play, and also lets you solve puzzles of various sizes and difficulties.

Requirements

File Format

The puzzle to be verified is specified in a text file that your main program will read in. This input file corresponds to the puzzle at the top of this writeup.

The first line of the file has two integers:

```
4 7
```

The first integer is the square dimension of the puzzle (4x4). The second integer indicates the number of regions in the puzzle, 7. The remainder of the file is dependent on these values.

The second line is empty, followed by 4 lines which represent the values in the rows for the potential solution. Notice how this matches the solution image above.

```
1 4 3 2
2 3 1 4
4 1 2 3
3 2 4 1
```

This will be followed by another blank line, followed by 7 lines for the regions. Each region has a target value (an integer) and a character for the math operation.

```
3 +
3 -
5 +
9 +
2 -
8 +
6 +
```

Assume the 7 regions are implicitly numbered from 0 (for "3 +"), to 6 (for "6 +"). We now need to specify what cells each region occupies. The last four lines of the file are used to indicate this.

```
0 3 3 3
0 4 4 5
1 1 6 5
2 2 6 5
```

Using (row, column) notation for the cells, this says that region 0, "3 +", is contained in the cells (0,0) and (1,0). Region 3, "9 +" is contained in the cells (0,1), (0,2) and (0,3).

Input File Guarantees

Please note, we guarantee that the input file is correct in its format. That means if the puzzle is 4x4, there will be 16 integer values for the potential solution and the region map. If it says there are 7 regions, there will be 7 of them with a single integer for the target value, and the operations will be one of the 4 strings: +, -, *, /.

In addition to these guarantees, you additionally do not need to check that each row and column in the potential solution contains the numbers 1 to N uniquely. It is the requirement of any input puzzle that it follows this rule.

Finally, we also guarantee that the region map will always be valid. It will only reference contiguous region numbers from 0 to N-1. Also, each region will contain at least two cells so that an operation can be performed.

Running the Program

The main class is run on the command line with the input file specified as an argument:

```
java KenKen filename
```

If the input file is not specified, you should display an error message and exit the program:

```
Usage: java KenKen filename
```

If the input file is specified, it is guaranteed to exist and will be valid. You don't have to error check the contents. Each region will be displayed, along with whether it is valid or not. For example, using the input file specified above, the output would be:

```
$ java KenKen input.1
Region: 0, target: 3, values: 1, 2, op: +, verify: true
Region: 1, target: 3, values: 4, 1, op: -, verify: true
Region: 2, target: 5, values: 3, 2, op: +, verify: true
Region: 3, target: 9, values: 4, 3, 2, op: +, verify: true
Region: 4, target: 2, values: 3, 1, op: -, verify: true
Region: 5, target: 8, values: 4, 3, 1, op: +, verify: true
Region: 6, target: 6, values: 2, 4, op: +, verify: true
```

If all the regions are valid, you should display this message and exit:

```
*** Puzzle is correct!
```

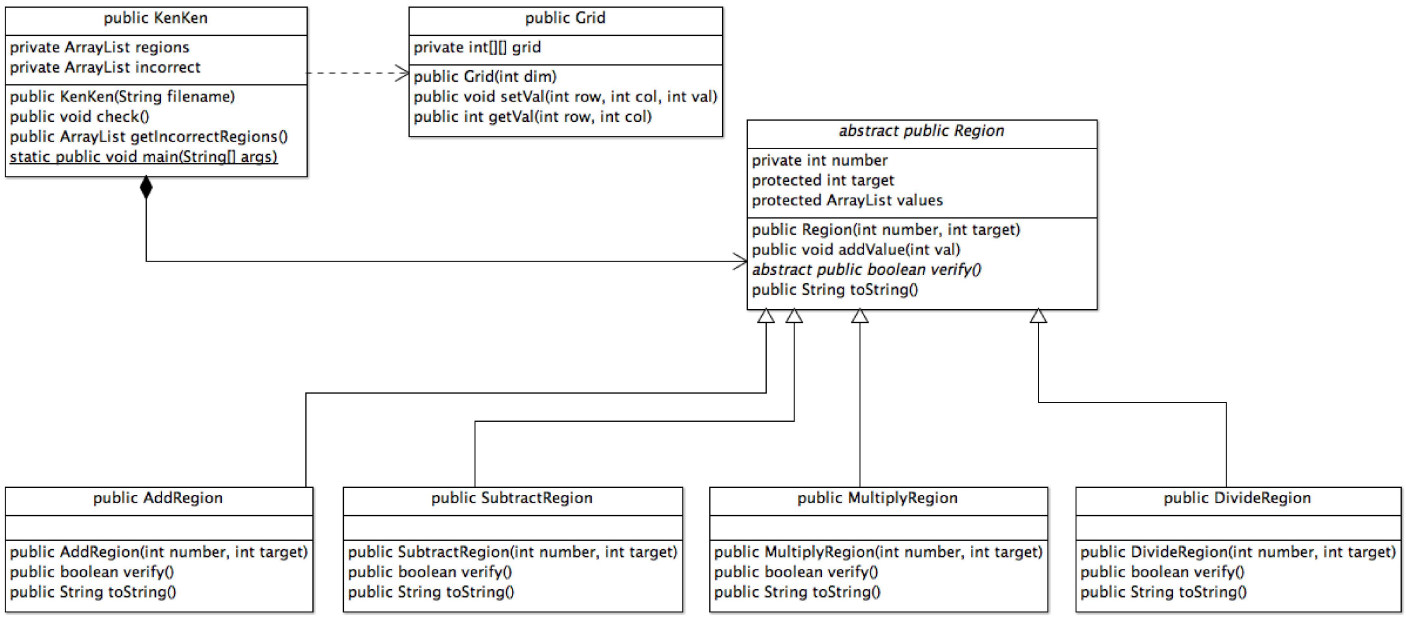
If any of the regions are invalid, you should display this message, with each invalid region number, and then exit:

```
*** Puzzle is incorrect! Invalid regions: [...]
```

Design

UML

This is the complete UML diagram with Java syntax. It includes the private state that we suggest you use.



Java Documentation

There are seven classes for this assignment.

- KenKen: This is the main class that reads in the puzzle, performs the checking and displaying the results.
- Grid: This is a class that is used in KenKen's constructor to store the solution to the puzzle.
- Region: This is the abstract class that all region types inherit from.
- AddRegion: This is a concrete subclass of Region that represents an addition region.
- SubtractRegion: This is a concrete subclass of Region that represents a subtraction region.
- MultiplyRegion: This is a concrete subclass of Region that represents a multiplication region.
- DivideRegion: This is a concrete subclass of Region that represents a division region.

Implementation

Reading the Input File

To read in the file in KenKen's constructor, you should use a Scanner that is attached to a File, e.g.:

```
Scanner in = new Scanner(new File(filename))
```

From here, you can use Scanner's nextInt() to read in integers, and Scanner's next() to read in the string for the region operation.

Because the file could not be present, you will need to handle the FileNotFoundException. To do this, you will need to declare that KenKen's constructor and main method both throw the exception.

```
import java.io.FileNotFoundException;

public KenKen(String filename) throws FileNotFoundException {...}
public static void main(String[] args) throws FileNotFoundException {...}
```

Grid

The puzzle solution is stored in a native 2-D array that is encapsulated by the Grid class. You will need to instantiate this in the Grid constructor.

```
this.grid = new int[dim][dim];
```

The Grid class is used temporarily by KenKen's constructor to read in the solution from the input file. Once the regions are created, and their locations are identified, the values from the grid get transferred into the internal array lists of Region's. At this point, the grid is no longer needed.

Testing

In addition to the input file from the top of the writeup, we are also providing a second one. It is a 5x5 puzzle with 9 regions. It uses all 4 of the operations.

Submission

Submit to try your source code and Git log:

```
$ try csapx-grd lab4-1 *.java log.txt
```

You can verify your submission with the command:

```
$ try -q csapx-grd lab4-1
```

Grading

The grade breakdown for the entire assignment is:

- Problem Solving: 15%
- In Lab Activity: 10%
- Functionality: 60%
- Design: -20%. Failure to follow the design outline will cause you to lose points for incorrectly using inheritance.
- Documentation - Code Format and Style: 10%
- GIT log: 5%