In The Name Of God

Puzzle Solver Project Notes

****

Student Name : Fateme Sharifi – Narges Habibi – Bahareh Fathi
Student ID Number: 8203593 – 8004311 – 8112199

****

1- Development Environment)

1-1 OS : Windows XP Professional
1-2 Editor : Eclipse V.3
1-3 Java : JDK 1.5

2- Usage Documentation)

For run program use:

- `> java Solve -h`

  For view a usage message.
- `>java Solve [-q] input`

Program can reads input data from 2-type source:

- Standard input, normally or by redirection(> java Solve -q < MyInput.txt)
- Several files. Sentences can be in some files and questions in others, etc.

In both mentioned types, input can be in free format .It means that each question or sentence can be in several lines with arbitrary numbers of white spaces. (\t, \n, \b)
But, must follow the given grammar exactly. (Otherwise, the "StringTokenizer" exception will be thrown!).

3- Overview of Coding Design)

3-1 Project contains 5-java file (5 class):
Solve, Input, Translator, PuzzleSolver, Output

3-2 The solve algorithm :
The puzzlesolver class take a cube with size of maximum number elements of (Names, Colors, Works), and set the whole square to one(1). Then according to positive sentences (Equals Vector) and negative sentences (nEquals Vector), multiplied the related squares by one or zero. Then, according to the rows of names, solve the puzzle. If there is one set square, it is answer and adds to Solve Vector and reset these work and color for other names. If there are more than one set square, take similar elements and add them to Solve Vector.

Class's definitions

Solve)

Methods: main (), start (), processCommandArgs ().
start (): This method first calls processCommandArgs () for processing command-line arguments. Next, creates objects form Translator, Output classes, and passes them to Input constructor, in addition, input file names Vector.
Finally, calls solvePuzzle () form Translator for solving puzzle and print result.

Input)

Methods: start (), read (), addToTranslatorOrOutput ().
Start (): If input file names Vector, contains no names, it calls read () with standard input. Otherwise, tries to open each file and calls read () on them.
read (): Processes input stream line-by-line and calls addToTranslatorOrOutput () for each complete sentence (question).If a line contains a non-complete sentence (question), holds it and continues for gather remained part.
AddToTranslatorOrOutput (): Receives a line with undetermined number of white spaces between its words. Then extracts words by using StringTokenizer class. Finally, add changed line (with one space between words) to Translator or Output.

Translator)

Methods: addSentences (), parseSentences (), solvePuzzle ().
ParseSentences (): According to given BNF grammar, extracts names, works, colors, is-relations and is-not-relations from each sentence and saves results in PuzzleSolver.
It assumes that sentences follow the grammar exactly.
solvePuzzle (): First calls parseSentences (), next calls cubeSolve () and solving () methods from PuzzleSolver and then parseQuestions () and printAnswer () from Output.

PuzzleSolver)

Methods: adder methods, get methods, private methods(max(), contain(), setDummy(), resteDummy()), cubeSolve(), solving().
cubeSolve():this method take a new cube with size of maximum number elements of (Names, Colors, Works), and set the whole square to one(1). Then according to positive sentences (that be added to Equals Vector) and negative sentences (that be added to nEquals Vector), multiplied the related squares by one(for positive sentence) or zero(for negative sentence).
solving():this method solve the puzzle according to the rows of names. If there is one set square, it is answer and adds (name, color, work) of this row to Solve Vector and reset these work and color for other names. If there are more than one set square, take

similar elements and add (name, color,) or (name, , work) to Solve Vector. If for one row whole squares are zero, that’s impossible.

Output)
Methods: addQuestion(), parseQuestions(), printAnswers().
parseQuestions(): According to given BNF grammar, extracts tokens of questions and find related name or color or work of Solve Vector(that give it from PuzzleSolver), and built correct answer, then add it to answers Vector.
It assumes that sentences follow the grammar exactly.
printAnswers(): According to entry order, print answers. If –q : only print answers, else, print a question and it’s answer and so.