CS61B, Fall 2012 **Project #1: Puzzle Solver**[*] **P. N. Hilfinger**

Due: Monday, 15 October 2012 at midnight

1 Background

You've probably seen those puzzles in which you are given clues such as "The plumber lives in the green house" and "Joe is not the archdeacon" and you are supposed to figure out everyone's occupation and taste in exterior latex enamel. In this project, you are to write a general-purpose solver for this type of problem.

The ground rules are that there is a row of houses, all of different colors, each occupied by one person. The people all have a single occupation (all different) and each lives in a single house. It is assumed that all questions and information in the input refer only to these people, houses, and occupations. It is also assumed that the numbers of houses, people, and occupations are equal to the number of specific houses mentioned, of specific people mentioned, or of specific occupations mentioned, whichever is largest. Thus, if a problem mentions only John, Mary, a plumber, a tailor, an architect, and a blue house, you may assume there are three people (one of whose names you don't know), and three houses (two of whose colors you don't know). For example, we want your program to take input such as this:

```
The plumber lives in the blue house.  Tom is not the carpenter.
John lives in the yellow house. John is not the carpenter.
Mary does not live in the blue house.
The architect does not live in the blue house.
What do you know about John?
What do you know about Mary?
Who is the plumber?
```

and print in response

```
John is the architect and lives in the yellow house.
Mary is the carpenter.
Tom is the plumber.
```

[I suggest you pause and figure out why these must be true under the ground rules.]

You can assume there will always be at least one question. Sometimes, the questions can't be answered. For example,

```
Jack lives in the white house.  Jack is the carpenter.
Jill is the clerk.  The carpenter lives in the yellow house.
What do you know about Jack?
```

to which the only response possible is

```
That's impossible.
```

Also, there can be several possible answers:

```
Jack lives in the blue house.  Mary does not live in the blue
house.  The mechanic lives in the red house.
There is a green house.
The architect lives around here. Who is the architect?
```

[*]This project is adapted from a problem in the 1996 Annual Berkeley Programming Contest.

to which we can only say

```
I don't know.
```

2 Input

The input consists of a set of sentences followed by a set of questions. Each sentence or question is on a single line, although there may be more than one sentence on a line. Otherwise, all input is (as in the examples) in *free format,* meaning simply that words are separated by any combination of blanks and tabs. Blank lines are ignored. The possible sentences and questions are as follows:

Sentence:
  Person-Designator House-Relation `the` *Color* `house.`
  Name Occupation-Relation `the` *Occupation*`.`
  Person-Designator `lives around here.`
  `There is a` *Color* `house.`
Question:
  `What do you know about` *Person-Designator*`?`
  `What do you know about the` *Color* `house?`
  `Who is the` *Occupation*`?`
  `Who lives in the` *Color* `house?`
  `What does the occupant of the` *Color* `house do?`
  `What does` *Name* `do?`
  `Where does` *Person-Designator* `live?`
Person-Designator:
  Name
  `the` *Occupation*
House-Relation:
  `lives in`
  `does not live in`
Occupation-Relation:
  `is`
  `is not`
Name:
  any capitalized word (alphabetic) other than a keyword
Occupation:
  any lower-case word (alphabetic) other than a keyword
Color:
  any lower-case word (alphabetic) other than a keyword

The notation above is a variety of *Backus-Naur Form (BNF),* and is described in Chapter 1 of *Assorted Materials on Java* in the reader. As you can see, the specification is very loose about what constitutes a color, occupation, or name, so that "Blah lives in the xyzzy house" or "The glorplefitzer lives in the green house" are perfectly acceptable. However, we don't allow any of the *keywords* (literal words that appear in the syntax) as a color, occupation, or name. Nobody, for example, can be named "Who," "There," or "The, " and "occupant" cannot be an occupation. Also, no word may be used for two different things: nobody works as a "green" if there is a green house. Words are separated by whitespace (blanks, tabs, newlines). The trailing question marks and periods appear immediately after their words (no space). The word "the" in *Person-Designator* is capitalized at the beginning of a sentence, both in the input and in the output (see §3).

Most of the sentences and questions should be self-explanatory. The last two kinds of sentence serve merely to introduce new names, occupations, or colors into the problem without saying anything about them. All such new terms must be mentioned in the sentences (before the first question), or the input is erroneous. That is, you can't use a name, color, or occupation for the first time in a question; the input

```
Mary is the carpenter.  Who lives in the red house?
```

is therefore illegal.

3 Output

First, your program should print out the original input, not including the questions, as a sequence of numbered sentences, each on one line In this echo of the original input, you should remove extra blanks, converting all stretches of whitespace to a single blank.

Following that, if the data are contradictory, then rather than answer any of the questions, just print out the line

```
That's impossible.
```

and stop.

Otherwise, print out each question on one line (again with whitespace compressed) followed by its answer on the next, as in this example:

```
Q: Who is the innkeeper?
A: Aloysius is the innkeeper.
```

Answers to questions have the forms shown in Table 1 (print out the appropriate choice from the responses listed).

Errors. If the input to your program is faulty, you must exit gracefully, using `System.exit(1)` to indicate that an error occurred. Our *only* other requirement in these situations is that you output a line containing the word "error" (in any combination of cases) on the *standard error output* (which is called `System.err` in Java and is otherwise just like `System.out`). In the absence of any errors in the input, exit using `System.exit(0)`.

4 Running the program

A user of your program starts it with a command having one of the following forms (things in square brackets are optional; '. . . ' means "one or more"):

```
java puzzle.Solve
java puzzle.Solve FILE
```

The first form just prints a concise, helpful description of the program and how to use it and then exits. The second uses the specified *FILE* as input.

5 What to Turn In

You will need to turn in `puzzle/Solve.java`, `puzzle/UnitTest.java` and any other source files you create.

Question	Possible Responses
`Who is the` *Occupation*`?`	*Name* `is the` *Occupation*`.`
	`I don't know.`
`Who lives in the` *Color* `house?`	*Name* `lives in the` *Color* `house.`
	`I don't know.`
`What does` *Name* `do?`	*Name* `is the` *Occupation*`.`
	`I don't know.`
`What does the occupant of the` *Color* `house do?`	`The` *Occupation* `lives in the` *Color* `house.`
	`I don't know.`
`Where does` *Person-Designator* `live?`	*Person-Designator* `lives in the` *color* `house.`
	`I don't know.`
`What do you know about` *Person-Designator*`?`	*Person-Designator* `lives in the` *Color* `house.`
	Name `is the` *Occupation*`.`
	Name `is the` *Occupation* `and lives in the` *Color* `house.`
	`Nothing.`
`What do you know about the` *Color* `house?`	*Person-Designator* `lives in the` *Color* `house.`
	Name `is the` *Occupation* `and lives in the` *Color* `house.`
	`Nothing.`

Table 1: Output formats.

This project will be evaluated in part on the thoroughness of your testing. Put JUnit test classes with names ending in `Test` (e.g., `SolveTest.java`) in your `puzzle` package, including one particular class named `UnitTest`, which should run *all* your individual JUnit tests as a *suite* (see the skeleton). Our autograder script will run `UnitTest` in the JUnit oframework, expecting it to pass. For black-box testing, we've set up simple scripts, much as we did for Project 0, that run test input files you supply and compare results with output files you supply. We've set up `gmake check` to run both sets of tests.

We will expect your finished programs to be workmanlike—consistently indented, well-commented, readably spaced. Don't leave debugging print statements lying around. In fact, don't use them; learn to use the debugger (either `gjdb` or that of `Eclipse` or your favorite Java-system vendor).

6 Advice

First, start thinking immediately! As with any complex problem, it is best to find a way to divide it into bite-sized pieces and to try to build the pieces separately.

You may be a bit curious as to how to solve such puzzles. I'm not going to give you huge sets of data, so ultimate efficiency is not the prime concern here. In fact, with the tests I'll give you, you should be able to get away with sheer brute force. First, make sure you have the same number of names, occupations, and house colors by creating "anonymous" ones. For each question in the input, try all assignments of house colors and occupations to people and find which ones are allowed by all the clues. From those assignments, extract all the (name, occupation, house color) triples that contain the name, occupation, or color that you are asked about. So, for "Who is the carpenter?", you would find all legal triples of the form (*some name,* 'the carpenter', *some color*). Then,

- If there are no such triples, the clues are contradictory and there is no solution.
- If all the triples have the same name and it isn't an "anonymous" one, then that is the unambiguous answer to any question asking for a name. Likewise for occupations and colors.
- Otherwise, you don't know.

Let me emphasize that this is just one way to go about it. You can get more ambitious (and efficient) if you want, but I don't require it (indeed, I don't recommend it unless you are pretty confident). In any case, *do not try to think about this problem in Java!* First, come up with a complete and rigorous pencil-and-paper procedure that you understand, and then figure out how to translate that into Java. As Prof. David Gries likes to say, "Program *into* your programming language, not *in* it."

Of course, even with an algorithm, there is a lot of structure needed to support it. It is useful to perform an *object analysis* in which you identify the kinds of things—the classes—you'll need to build. For example, from the description of the problem and suggested algorithm, you might consider these:

There are numerous classes you might find useful, including:

- `List` and `ArrayList` or `LinkedList`.
- `Set` and `HashSet` or `TreeSet`.
- The classes `java.util.Scanner` and `java.util.regex.Pattern` are described in chapter 4 of *A Java Reference* in the class reader. You will probably be most interested in the `Scanner` methods `findInLine`, `nextLine`, `hasNextLine`, and `match`. I suggest *not* using the other `next...` and `has...` methods for this assignment. Properly used, these make the task of reading questions and answers *really easy.*. (I mean that; you should *not* be spending any great amount of time on input.)
- The class `java.io.File` stands for a file name, and is useful as an argument to `Scanner`'s constructor.

Above all, it is always fair to ask for help and advice. We don't *ever* want to hear about how you've been beating your head against the wall over some problem for hours. If you can't make progress, don't waste your time guessing: ask.

7 Examples

Input File `inp1`**:**

```
Sue lives around here.  There is a brown house.
The professor lives around here.
What do you know about Sue?
```

Output after the command `java puzzle.Solve inp1`**:**

```
1. Sue lives around here.
2. There is a brown house.
3. The professor lives around here.

Q: What do you know about Sue?
A: Sue is the professor and lives in the brown house.
```

Input File `inp2`**:**

```
John is not the carpenter.
The plumber lives in the blue       house.
John lives in the yellow house.
Mary does not live in the blue house.  Tom lives around here.
The architect lives around here.
What do you know about John? What do you know about Mary?
What do you know about Tom?
```

Output after the command `java puzzle.Solve inp2`**:**

```
1. John is not the carpenter.
2. The plumber lives in the blue house.
3. John lives in the yellow house.
4. Mary does not live in the blue house.
5. Tom lives around here.
6. The architect lives around here.

Q: What do you know about John?
A: John is the architect and lives in the yellow house.
Q: What do you know about Mary?
A: Mary is the carpenter.
Q: What do you know about Tom?
A: Tom is the plumber and lives in the blue house.
```

Input File `inp3`**:**

```
Jack lives in the blue house.  Mary does not live in the blue house.
The mechanic lives around here.
There is a red house.  The architect lives around here.
The sailor lives around here.
Who is the mechanic?   What do you know about Jack?
What do you know about Mary?
```

Output after the command `java puzzle.Solve inp3`**:**

```
1. Jack lives in the blue house.
2. Mary does not live in the blue house.
3. The mechanic lives around here.
4. There is a red house.
5. The architect lives around here.
6. The sailor lives around here.

Q: Who is the mechanic?
A: I don't know.
Q: What do you know about Jack?
A: Jack lives in the blue house.
Q: What do you know about Mary?
A: Nothing.
```

Input File `inp4`**:**

```
Jack lives in the white house.
Jack is the carpenter.
Jill is the clerk.
The carpenter lives in the yellow house.
Who is the clerk?
What do you know about Jill?
```

Output from the command `java puzzle.Solve inp4`**:**

```
1. Jack lives in the white house.
2. Jack is the carpenter.
3. Jill is the clerk.
4. The carpenter lives in the yellow house.

That's impossible.
```