In Lab 1, you will complete a program for solving KenKen puzzles. The initial program is kenken.zip, which you can download from the course web site. This will be done by implementing KenKen as a CSP (see Chapter 4 of the textbook). The initial program already has code to perform arc consistency and domain splitting. You will need to add code to create a CSP (a list of constraints and a list of variables) and to specify constraints.

Initial code is provided for you on the course web site. Your grade on the lab will depend on the performance of your program when I run it on test problems. Your submission should also include a short description of how your program performs.

**KenKen**


This puzzle has eight *cages*. A cage specifies a target value and an operation. A + operation means that the sum of the values in the cage must be equal to the target. A − operation means that the the difference between the two values (in one order or the other) must be equal to the target. A × operation means that the product of the values in the cage
must be equal to the target. A ÷ operation means that quotient of the two values (in one order or the other) is equal to the target. Any cage with just one cell means that the value must be equal to the target.

puzzles.txt in the download represents this puzzle in plain text as follows:

```
4 8
1 - 2 0 0 1 0
7 + 2 0 1 1 1
1 = 1 0 2
2 / 2 0 3 1 3
48 * 3 1 2 2 2 2 3
3 + 2 2 0 2 1
4 = 1 3 0
6 + 3 3 1 3 2 3 3
```

The first line specifies a 4 × 4 puzzle with 8 cages. Each of the following lines specifies a cage: the target value, the operation, the number of cells, and the coordinates of each cell.

**Environment**

The environment is defined in KenKenEnvironment.java. It reads the puzzles from puzzles.txt and sends them to the agent. The environment expects to read a solution from the agent as a single line listing the value for each cell. The KenKenEnvironment then checks the result and prints success or failure to the console.

When there are no more puzzles, KenKenEnvironment prints quit and finishes.

**Interact**

Interact.java is used to connect KenKenEnvironment to a KenKen agent. It does this by running the environment and the agent in different threads. Both the environment and the agent need to implement the Runnable interface.

Interact.java creates Java pipes so that the environment and agent have their own input and output streams. It monitors these streams so that any line output by one will be input to the other. Interact.java also prints to the console anything output by the environment and agent. The environment and agent can use System.out to print anything to the console.

The picknumber program is an example of how Interact.java works with an environment and an agent. You can download picknumber.zip from the course web site.

**Agent**

Your task is to complete an agent program (KenKenSolver.java) to interact with the KenKen environment. Your program should read in a puzzle, find and print out a solution, and repeat. KenKenSolver.java is already coded to read in puzzles and quit when it is supposed to, but it does not print out solutions.

The way KenKenSolver.java should find a solution is by formulating the problem as a *constraint satisfaction problem* (see Chapter 4 of the textbook), and applying the *generalized*
arc consistency algorithm (see Section 4.5) and domain splitting (see Section 4.6). Domain splitting is a form of depth-first search (see Section 3.5.1). The download contains most of the code for consistency and domain splitting.

Solving the KenKen puzzles by any other method will earn a zero for this assignment.

CSPTest.java provides very simple examples of CSPs and shows the result of the GAC algorithm (implemented by the consistency method). Note that the GAC algorithm is distributed over the Constraints, that is, each Constraint applies arc consistency to its Variables. This will be covered in more detail in class.

Constraint Subclasses

The download contains code for ConstraintNotEqual.java and ConstraintEqual.java. Both of them are subclasses of Constraint, which does the heavy-duty work for generalized arc consistency and domain splitting.

A ConstraintNotEqual constraint is intended for two variables that should not be equal to each other. A ConstraintEqual constraint is intended for cages with just one variable to ensure that the variable is equal to the target. Note how each of these classes are defined.

You will need to add four additional classes: ConstraintPlus, ConstraintMinus, ConstraintTimes, and ConstraintDivide. Each class will be a subclass of Constraint. Each class will have a target field, a constructor, and a test method. Note that specific values are passed to the test method. The code in Constraint.java ensures that every possible combination of values will be tested.

For a KenKen puzzle, a CSP needs to have one Variable for each cell. For example, a 4 × 4 puzzle requires 16 Variables, and a 6 × 6 puzzle requires 36 Variables.

createCSP method

You will need to code the createCSP method in KenKenSolver.java.

A KenKen object is passed to the createCSP method. This object is created when the KenKen.readPuzzle method reads in the puzzle. Look at KenKen.java, Cage.java, and Cell.java to see how the puzzle is represented and how you can obtain the information you need.

For a KenKen puzzle, a CSP needs to have one Variable for each cell. For example, a 4 × 4 puzzle requires 16 Variables, and a 6 × 6 puzzle requires 36 Variables. The getSize method returns the size of the puzzle.

For each pair of Variables in the same row or column, a ConstraintNotEqual constraint must be created and added to the CSP. An n × n puzzle will have n²(n − 1) Constraint-NotEqual constraints.

There needs to be a constraint for each cage. The cages of a KenKen object can be obtained using the getCage method (the getNumCages method returns the number of cages). The target and operation of a Cage object can be obtained using the getTarget and getOperation methods. The cells of a Cage object can be obtained using the getCell method (the getNumCells method returns the number of cells). You will need a way to associate each cell with a Variable.
Turning in Your Lab

Submit the folder containing all your Java files to Blackboard. Your program should run under the following conditions: after all the .class files are deleted, then:

```
javac Interact.java
java Interact
```

should solve the puzzles. You may also submit your program as an Eclipse project.

The score of your lab will primarily depend on the performance of the program. In addition to the puzzles in puzzles.txt, your program will be tested on an additional 10 puzzles. Some will be smaller than $4 \times 4$, some larger than $6 \times 6$. Your score will be the percentage of puzzles that you solve.