AP Computer Science
Curriculum Module: Inheritance and Polymorphism with Sudoku

Richard Kick
Newbury Park High School
Newbury Park, CA
Moorpark College
Moorpark, CA

Inheritance and Polymorphism with Sudoku

Object-oriented programmers agree that using objects can be easy, but designing effective object-oriented programs is relatively difficult. It is like cabinet making from years gone by: the wood-crafting artistry required to build beautiful and equally functional cabinets took years of practice and study with master craftsmen. Yet using the cabinets produced by these artists was, and is, easy. Today, well-designed class libraries are easy to read, understand, and use for the creation of new programs. But the design of these easy-to-use libraries requires the code craftsmen of today. It is our hope that our students will begin their move toward the mastery of code production by learning the important concepts related to the field of object-oriented programming.

This article will discuss inheritance and polymorphism as they relate to the Java interface. Inheritance is a powerful mechanism that allows programmers to create new code from existing code by building upon shared attributes and functionality identified among related program concepts. The abstraction of important program concepts allows for safe modification of programs that require fewer code changes. We will explore inheritance using the Java interface mechanism by looking at a particular programming problem: the Sudoku board game.

Sudoku is a game that requires players to determine missing digits in a 9 by 9 grid until all 81 values are revealed. The digits used in the game are the consecutive integers 1 through 9. The grid has logical groups of numbers that must contain each of the 9 possible digits without repetition. These logical groups consist of the nine rows, nine columns, and nine 3 by 3 grids, know as regions, as shown below. All of these groups will be referred to by the term **section**.

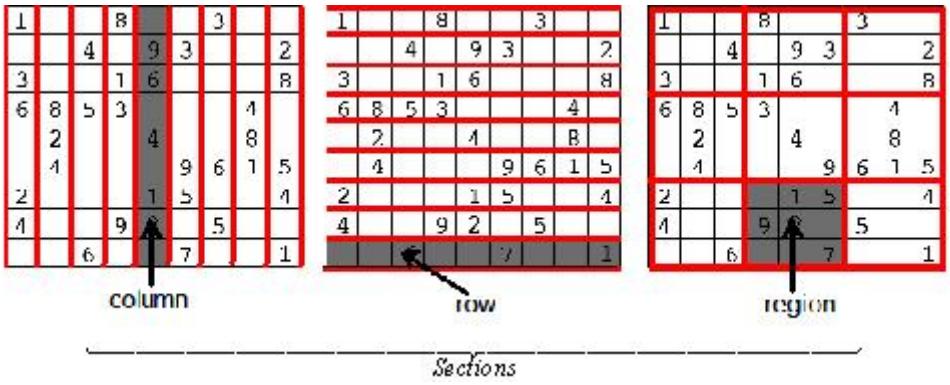


Figure 0

The game begins with some of the 81 numbers already displayed, as shown in figure 0. Games that are categorized as “easy” games usually have several squares that have only one possible digit based on other digits that are in the remainder of the corresponding sections (row, column, or region). Many Web-based versions of Sudoku have all possible numbers for each square displayed. The game progresses by having the player eliminate values that are not possible until only one value per square remains. This technique is used for the displays in figures 1 and 2.

If a square has a value, say 1, then all sections that contain that square may not include any other squares that have a 1, as illustrated in figure 1. More difficult games require that more subtle considerations be taken into account in order to determine the values that should occupy the unfilled squares. One of the more subtle Sudoku strategies is illustrated in figure 2. A section that contains a common pair of possible values can be used to eliminate both of those digits in the remainder of its section. Notice in the second row, the values 5 and 6 are possible values for the second and eighth square in that row. This means that neither a 5 nor a 6 can occur in any other square in that row. This logic allows the Sudoku player to eliminate the 5 in the first square in row 2, leaving the number 8 in that square. Similar elimination can be performed on the first and seventh rows, the seventh column, and the bottom right region.

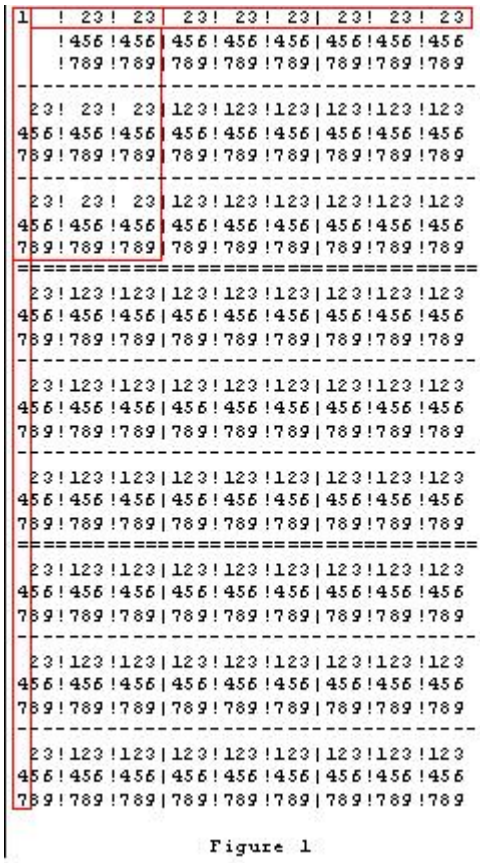


Figure 1

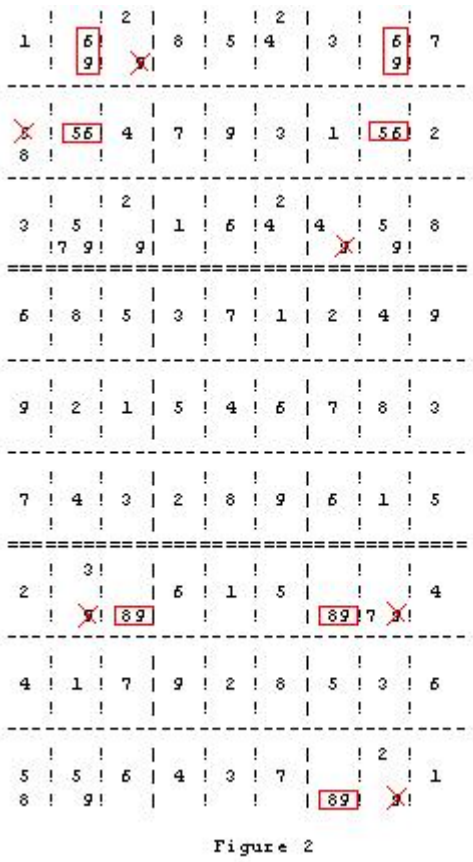


Figure 2

In order to model the game of Sudoku, we must identify important concepts related to the game. This identification process can be accomplished through conversation, scenario writing, or simply writing a detailed description of what occurs while playing the game. The nouns generated by these communications are candidates for classes. Verbs can be used as candidates for methods. Multiple iterations of this process, combined with effective decision making by experienced OOP programmers, will result in a list of classes and methods for the game.

We shall discuss one particularly important concept related to the game of Sudoku and model it as a class: the `Board` class. The `Board` class must model the squares in which the integer values reside, including the sections of the board that contain multiple squares. The `Board` must also provide the functionality that allows the player to modify the integers as they discover the specific values for each square on the board.

It is common for introductory-level students to begin writing programs by placing all, or nearly all, of their code into a single class. As projects grow in complexity and scope, the need to divide programs into logical components that are understandable and manageable becomes clear. This separation of code into smaller, more specialized blocks allows for easier maintenance and extensibility.

When designing the program for Sudoku, we should consider how altering the rules of the game would require the code to be changed. In particular, what if new games required board squares to be updated in different ways when a value is placed on the board? How can the amount of required code modification be minimized when writing games that are defined in this way?

To illustrate the use of the Java interface and the power of polymorphism, we will examine an implementation of a `Board` class that abstracts the sections of the board into classes that all implement the same interface. This will allow the `Board` class to process the sections in the same way, letting each specific implementation of the `Section` interface carry out the unique details of each particular implementation.

Looking at the code segments below, notice the board has sections that must be initialized in the constructor. Defining `Row`, `Column`, and `Region` classes that implement the `Section` interface allows the `Board` to process all sections in the same way, leaving the processing details to the particular implementations of `Section`. This also allows for modifications of the game through the definition of other sections that process values in their own way, without changing the `Board` class.

```
/**
 * Board class models the 9 by 9 rectangular array of squares.
 * Each square can be blank or have an integer value 1 through 9.
 * Each square also has a list of possible values that may be used
 * for that square's value.
 */
public class Board implements Sudoku
{
   // all squares on the Sudoku board
   private Square[][] square;
   // true if a square on the board has a new state; false otherwise
   private boolean stateChanged;
   // all sections that are defined for the board
   private Section[] section;
```

```
  /**
   * Initialize all data related to the board
   */
  public Board()
  {
    stateChanged = false; // indicates if a square has been changed

    square = new Square[NUM_ROWS][NUM_COLUMNS];

    for (int r = 0; r < square.length; r++)
    {
      for (int c = 0; c < square[0].length; c++)
      {
        // initializing a square with 0 means it has an
        // undetermined value (possible values are 1 to 9)
        square[r][c] = new Square(0, r, c);
      }
    }
    sectionsInit();
  }

  /*
   * Initialize all sections (rows, columns, and regions) so each
   * type of section is initialized with consecutive integer values
   * beginning with 0.
   */
  private void sectionsInit()
  {

   // initialize all sections (rows, columns, regions)
   // <... code omitted ...>

  }

  // ... additional code ommitted
}

/**
 * Represents a portion of the board that must be
 * updated when a board square contained within this
 * section is modified
 */
public interface Section
{
      /**
       * @param  square is the board data that holds all Sudoku information
       * @param  aSquare is the Square used to update all Section values
       */
      void update(Square[][] square, Square aSquare);

}
```

```
/**
 * Row represents a row of squares on a Sudoku board
 */
public class Row implements Section
{
    private int rowNum;

    public Row(int num)
    {
      rowNum = num;
    }

    public void update(Square[][] square, Square aSquare)
    {
      if ( aSquare.getRow() == rowNum )
      {
        for (int item = 0; item < square.length; item++)
        {
         if ( square[rowNum][item].getValue() == 0 )
            square[rowNum][item].removePossibleValue( aSquare.getValue() );
        }
      }
    }
}
```

Exercise 1

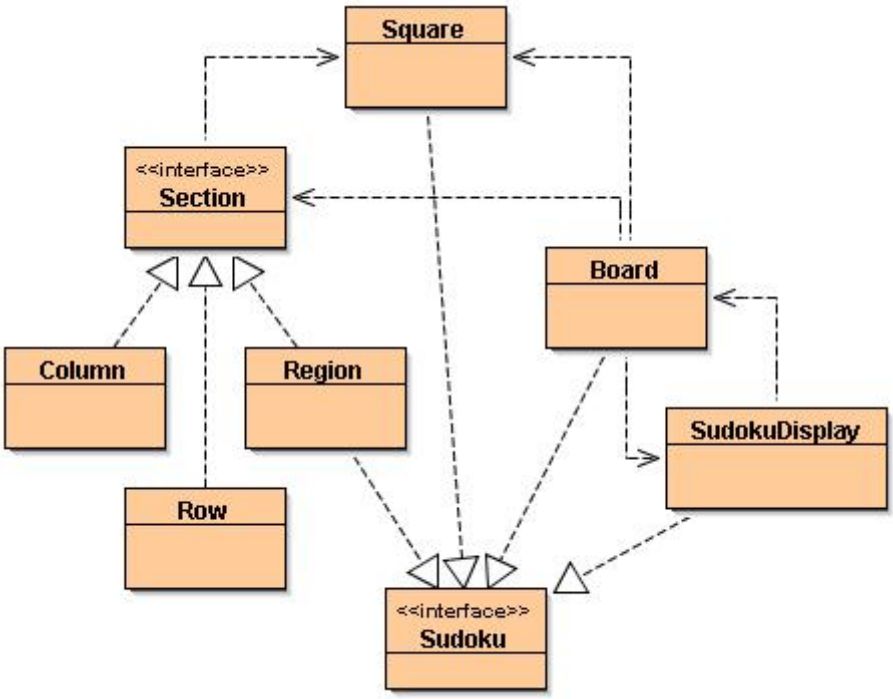
Given the `Row` class, write the `Column` and `Region` classes.

Exercise 2

Write the code for the private `Board` method `sectionsInit` that initializes the `Row`, `Column`, and `Region` sections of the board.

Exercise 3

Using the UML diagram below, and given that `Sudoku` is an interface that defines constants for use throughout the Sudoku program, describe the relationships between the classes for this program. Discuss the advantages and disadvantages of using this design for the Sudoku program.



Exercise 4

Modify the `Section` interface to include the following methods.

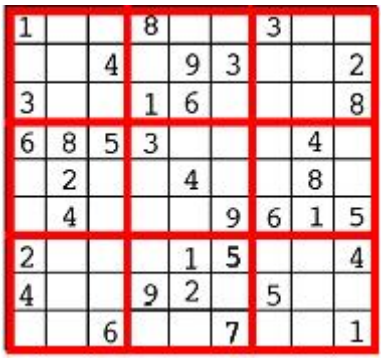
```
/**
 * @return array of int of length two that contains the first pair of
 * possible values that occur in two squares of this section;
 * returns null if no such pair exists
 */
  int[] getPair(Square[][] square);

/**
 * @param array of int of length two that contains a pair of
 * possible values; removes the two values from all squares not
 * having the pair as their only possible values
 */
  void processPair(Square[][] square, int[] pair);
```

Implement these methods in the `Row` class. Given all other classes, run the entire Sudoku program and solve the puzzle whose initial state is given by

```
Board sudoku = new Board();
SudokuDisplay display = new SudokuDisplay(sudoku);
```

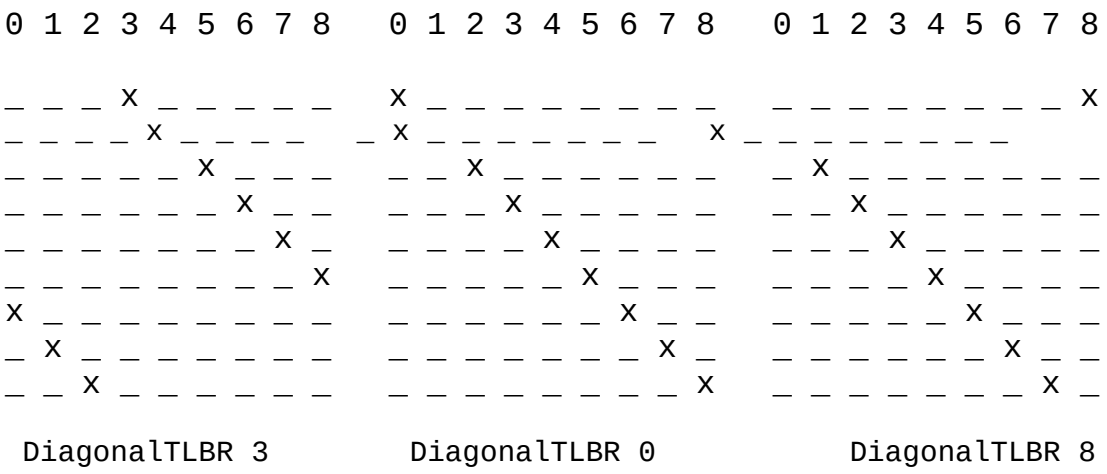
```
sudoku.updateSections(0, 0, 1);
sudoku.updateSections(0, 3, 8);
sudoku.updateSections(0, 6, 3);
sudoku.updateSections(1, 2, 4);
sudoku.updateSections(1, 4, 9);
sudoku.updateSections(1, 5, 3);
sudoku.updateSections(1, 8, 2);
sudoku.updateSections(2, 0, 3);
sudoku.updateSections(2, 3, 1);
sudoku.updateSections(2, 4, 6);
sudoku.updateSections(2, 8, 8);
sudoku.updateSections(3, 0, 6);
sudoku.updateSections(3, 1, 8);
sudoku.updateSections(3, 2, 5);
sudoku.updateSections(3, 3, 3);
sudoku.updateSections(3, 7, 4);
sudoku.updateSections(4, 1, 2);
sudoku.updateSections(4, 4, 4);
sudoku.updateSections(4, 7, 8);
sudoku.updateSections(5, 1, 4);
sudoku.updateSections(5, 5, 9);
sudoku.updateSections(5, 6, 6);
sudoku.updateSections(5, 7, 1);
sudoku.updateSections(5, 8, 5);
sudoku.updateSections(6, 0, 2);
sudoku.updateSections(6, 4, 1);
sudoku.updateSections(6, 5, 5);
sudoku.updateSections(6, 8, 4);
sudoku.updateSections(7, 0, 4);
sudoku.updateSections(7, 3, 9);
sudoku.updateSections(7, 4, 2);
sudoku.updateSections(7, 6, 5);
```

```
sudoku.updateSections(8, 2, 6);
sudoku.updateSections(8, 5, 7);
sudoku.updateSections(8, 8, 1);
```

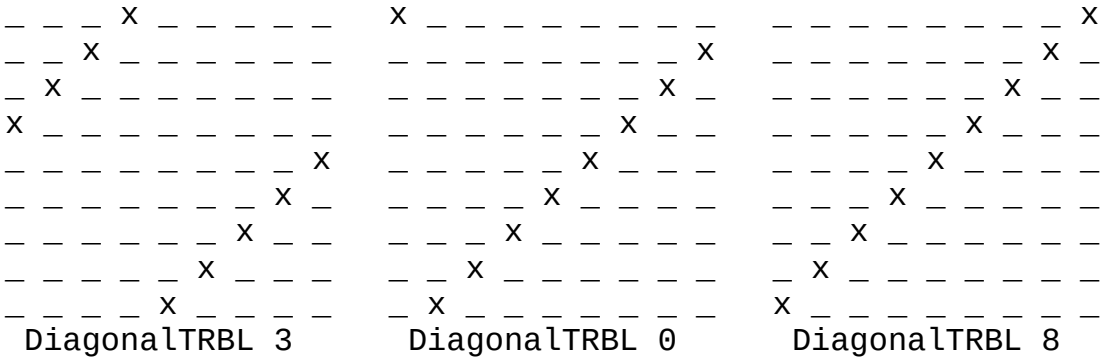


Exercise 5

Replace the `Column` and `Region` classes with classes called `DiagonalTLBR` and `DiagonalTRBL` that implement `Section`. `DiagonalTLBR` represents a collection of 9 squares on a diagonal. The positions of the squares for this class can be followed top to bottom, left to right, wrapping from the right edge to the left edge and from the bottom edge to the top edge as needed. The `DiagonalTRBL` class is similar with the squares extending from top right to bottom left. Each diagonal is numbered with an integer (`diagNum`) between 0 and 8 corresponding to the column number of the square in row 0.



Note: `DiagonalTLBR 0` includes only the main diagonal. For any square that is contained in `row`, `column`, `diagNum = (NUM_DIGITS + column - row) % NUM_DIGITS`



Note: `DiagonalTRBL 8` includes no break in the diagonal. For any square that is contained in `row`, `column`, `diagNum = (column + row) % NUM_DIGITS;`

For each value placed in a square, have the `DiagonalTLBR` update method remove all possible values from the diagonal (which extends from the top left to the bottom right of the board) that include the modified square. The diagonal should extend to the edges of the board and wrap around to include one square from each row and one square from each column. The `DiagonalTRBL` is similar.

Exercise 6

Rewrite the code for the private `Board` method `sectionsInit` that initializes the `Row`, `DiagonalTLBR`, and `DiagonalTRBL` sections of the board. Run the `DiagonalTLBRTestMain` to test your code.

A Sudoku Programming Project

In my AP Computer Science AB course (this should also work with an A course), I had my students gather into groups of three and discuss strategies for playing Sudoku. After a class period of playing the game, I instructed them to discuss in their small groups possible designs for a program that would solve any Sodoku puzzle. I suggested that they consider the data structures, algorithms, and the user interface for input and output. They were instructed to divide the work among group members, using the strengths of each member to maximize the quality of the resulting code. I directed them to work on the data structures and algorithm efficiencies, documentation, user interfaces, and testing.

Finally, a competition between groups was defined. Each group was given a Sudoku puzzle in the form of a projection on a screen at the front of the classroom. They were then given a total of three minutes to enter the puzzle, solve the puzzle, and present the solution to the judge (the teacher) in the form of a 9 by 9 grid of digits. The judge checked the solutions by a quick scan of the entire grid, followed by a more careful check of a particular row, column, or region that was strategically chosen before the competition.

Students were given a week (including one weekend) to complete their team program. On the day of the competition, students submitted their documented code as attachments to email messages sent to the teacher. They were then given puzzles one at a time, starting with two relatively easy puzzles, followed by two medium-difficulty puzzles, and finally, two puzzles that were very difficult. As a tiebreaker, students were asked to use their software to generate a Sudoku puzzle that would be presented to other groups. One group generated a puzzle that no other group could solve quickly.

This project was a success because students learned that there are many important aspects to software creation besides typing code. They learned the importance of responsibility distribution and teamwork. They also enjoyed the experience of working together toward a common goal. Each group found success in solving several puzzles in various ways. One group ran simple algorithms and used hand-entry techniques to complete partial solutions. Other groups explored techniques that involved computer guesses at difficult times in the solving process, backtracking if the guess did not result in a solution, then trying other guesses until a solution was found. Finally, one group explored the “dancing links” algorithm by Knuth. This algorithm involves relatively high-level linear algebra and the use of sparse matrices. This experience allowed them to find a deeper appreciation for the power of mathematics for complex problem solving.

I certainly recommend this experience to other computer science teachers and students. Sudoku is a fascinating, enjoyable, and sometimes addictive game. Students and teachers can enjoy both the programming and the game play. The game seems to be gender neutral, but my students were all male. I look forward to trying this project out with a more diverse group in the future.

Possible Extensions

Create iterators for each type of section and rewrite the `Row`, `Column`, `Region`, and `Diagonals` classes using iterators.

Look for common, repeated code and create methods that centralize the code to reduce the potential for the introduction of bugs during code modification.

Sudoku Step by Step

```
sudoku.updateSections(0, 0, 1);
```

```
123!123!123|123!123!123|123!123!123        ! 23! 23| 23! 23! 23| 23! 23! 23
456!456!456|456!456!456|456!456!456       1 !456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789        !789!789|789!789!789|789!789!789
-----------------------------------       -----------------------------------
123!123!123|123!123!123|123!123!123        23! 23! 23|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
-----------------------------------       -----------------------------------
123!123!123|123!123!123|123!123!123        23! 23! 23|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
===================================       ===================================
123!123!123|123!123!123|123!123!123        23!123!123|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
-----------------------------------       -----------------------------------
123!123!123|123!123!123|123!123!123        23!123!123|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
-----------------------------------       -----------------------------------
123!123!123|123!123!123|123!123!123        23!123!123|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
===================================       ===================================
123!123!123|123!123!123|123!123!123        23!123!123|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
-----------------------------------       -----------------------------------
123!123!123|123!123!123|123!123!123        23!123!123|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
-----------------------------------       -----------------------------------
123!123!123|123!123!123|123!123!123        23!123!123|123!123!123|123!123!123
456!456!456|456!456!456|456!456!456       456!456!456|456!456!456|456!456!456
789!789!789|789!789!789|789!789!789       789!789!789|789!789!789|789!789!789
```

```
sudoku.updateSections(0, 3, 8);
sudoku.updateSections(1, 2, 4);
```

```
sudoku.updateSections(0, 6, 3);
sudoku.updateSections(1, 4, 9);
```

```
   ! 2 ! 2 |    ! 2 ! 2 |    ! 2 ! 2
 1 ! 56! 56| 8 !456!456| 3 !456!456
   !7 9!7 9|    !7 9!7 9|    !7 9!7 9
-----------------------------------
 23! 23!   |123!123!123|12 !12 !12
 56! 56! 4 | 56! 56! 56| 56! 56! 56
789!789!   |7 9!7 9!7 9|789!789!789
-----------------------------------
 23! 23! 23|123!123!123|12 !12 !12
 56! 56! 56|456!456!456|456!456!456
789!789!789|7 9!7 9!7 9|789!789!789
===================================
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!789!789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!789!789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!789!789|789!789!789
===================================
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!789!789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!789!789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!789!789|789!789!789
```

```
   ! 2 ! 2 |    ! 2 ! 2 |    ! 2 ! 2
 1 ! 56! 56| 8 !456!456| 3 !456!456
   !7 9!7 9|    !7  !7  |    !7 9!7 9
-----------------------------------
 23! 23!   |123!   !123|12 !12 !12
 56! 56! 4 | 56! 9 ! 56| 56! 56! 56
78 !78 !   |7  !   !7  |78 !78 !78
-----------------------------------
 23! 23! 23|123!123!123|12 !12 !12
 56! 56! 56|456!456!456|456!456!456
789!789!789|7  !7  !7  |789!789!789
===================================
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!78 !789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!78 !789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!78 !789|789!789!789
===================================
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!78 !789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!78 !789|789!789!789
-----------------------------------
 23!123!123|123!123!123|12 !123!123
456!456! 56|456!456!456|456!456!456
789!789!789|7 9!78 !789|789!789!789
```

```
sudoku.updateSections(1, 5, 3);
sudoku.updateSections(2, 0, 3);
sudoku.updateSections(2, 4, 6);
```

```
sudoku.updateSections(1, 8, 2);
sudoku.updateSections(2, 3, 1);
sudoku.updateSections(2, 8, 8);
```

```
   ! 2 ! 2 |    ! 2 ! 2 |    !   !
 1 ! 56! 56| 8 !45 !45  | 3 !456!456
   !7 9!7 9|    !7  !7  |    !7 9!7 9
-----------------------------------
   !   !   |    !   !   |1  !1  !
 56! 56! 4 | 5 ! 9 ! 3  | 56! 56! 2
78 !78 !    |7  !   !    |78 !78 !
-----------------------------------
   ! 2 ! 2 |    !   ! 2 |    !   !
 3 ! 5 ! 5 | 1 ! 6 !45  |45 !45 !45
   !789!789|    !   !7  |789!789!789
===================================
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!789
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!789
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!789
===================================
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!789
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!789
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!789
```

```
   ! 2 ! 2 |    ! 2 ! 2 |    !   !
 1 ! 56! 56| 8 !45 !45  | 3 !456!456
   !7 9!7 9|    !7  !7  |    !7 9!7 9
-----------------------------------
   !   !   |    !   !   |1  !1  !
 56! 56! 4 | 5 ! 9 ! 3  | 56! 56! 2
78 !78 !    |7  !   !    |7  !7  !
-----------------------------------
   ! 2 ! 2 |    !   ! 2 |    !   !
 3 ! 5 ! 5 | 1 ! 6 !45  |45 !45 ! 8
   !7 9!7 9|    !   !7  |7 9!7 9!
===================================
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!7 9
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!7 9
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!7 9
===================================
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!7 9
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!7 9
-----------------------------------
 2 !123!123|  23!123!12 |12 !123!1 3
456!456! 56|456!45 !456 |456!456!456
789!789!789|7 9!78 !789 |789!789!7 9
```

```
sudoku.updateSections(3, 0, 6);
sudoku.updateSections(3, 2, 5);
sudoku.updateSections(3, 7, 4);
sudoku.updateSections(4, 4, 4);
```

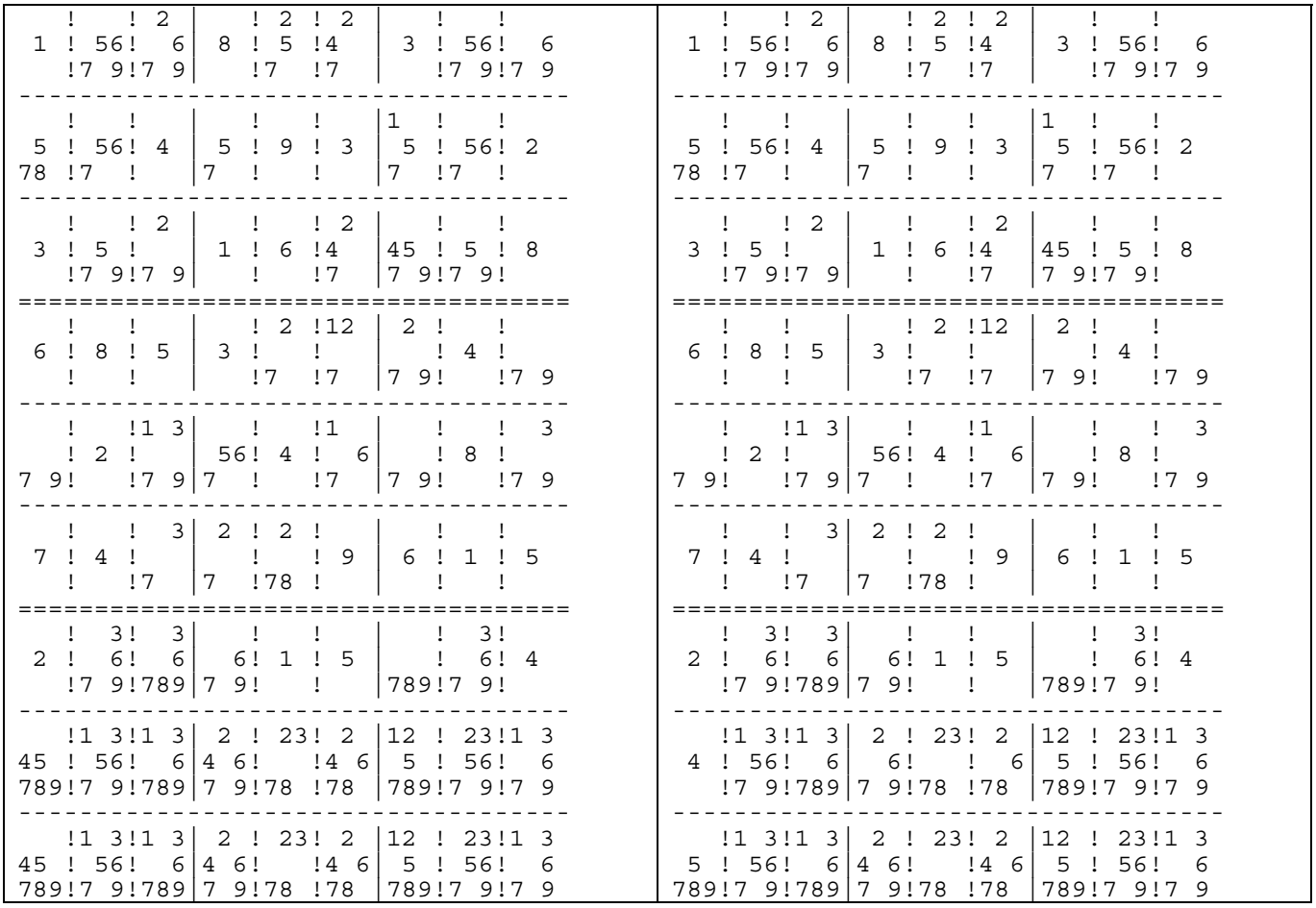
```
sudoku.updateSections(3, 1, 8);
sudoku.updateSections(3, 3, 3);
sudoku.updateSections(4, 1, 2);
sudoku.updateSections(4, 7, 8);
```

```
   !   ! 2 |   ! 2 ! 2 |   !   !
 1 ! 56!  6| 8 ! 5 !45 | 3 ! 56!456
   !7 9!7 9|   !7  !7  |   !7 9!7 9
-----------------------------------
   !   !   |   !   !   |1  !1  !
 5 ! 56! 4 | 5 ! 9 ! 3 | 56! 56! 2
78 !7  !   |7  !   !   |7  !7  !
-----------------------------------
   !   ! 2 |   !   ! 2 |   !   !
 3 ! 5 !   | 1 ! 6 !45 |45 ! 5 ! 8
   !7 9!7 9|   !   !7  |7 9!7 9!
===================================
   !   !   |   !12 !12 |12 !   !1
 6 ! 8 ! 5 | 3 !   !   |   ! 4 !
   !   !   |   !7  !7 9|7 9!   !7 9
-----------------------------------
   !   !1 3|   !   !1  |1  !1 3!1 3
   ! 2 !   | 56! 4 ! 56| 56! 56! 56
7 9!   !7 9|7 9!   !789|789!789!7 9
-----------------------------------
   !1 3!1 3| 2 !12 !12 |12 !123!1 3
4  !4  !   | 56! 5 ! 56| 56! 56! 56
7 9!7 9!7 9|7 9!78 !789|789!789!7 9
===================================
 2 !1 3!123| 2 !123!12 |12 !123!1 3
45 !456!  6|456! 5 !456|456! 56!456
789!7 9!789|7 9!78 !789|789!789!7 9
-----------------------------------
 2 !1 3!123| 2 !123!12 |12 !123!1 3
45 !456!  6|456! 5 !456|456! 56!456
789!7 9!789|7 9!78 !789|789!789!7 9
-----------------------------------
 2 !1 3!123| 2 !123!12 |12 !123!1 3
45 !456!  6|456! 5 !456|456! 56!456
789!7 9!789|7 9!78 !789|789!789!7 9
```

```
   !   ! 2 |   ! 2 ! 2 |   !   !
 1 ! 56!  6| 8 ! 5 !45 | 3 ! 56!456
   !7 9!7 9|   !7  !7  |   !7 9!7 9
-----------------------------------
   !   !   |   !   !   |1  !1  !
 5 ! 56! 4 | 5 ! 9 ! 3 | 56! 56! 2
78 !7  !   |7  !   !   |7  !7  !
-----------------------------------
   !   ! 2 |   !   ! 2 |   !   !
 3 ! 5 !   | 1 ! 6 !45 |45 ! 5 ! 8
   !7 9!7 9|   !   !7  |7 9!7 9!
===================================
   !   !   |   !12 !12 |12 !   !1
 6 ! 8 ! 5 | 3 !   !   |   ! 4 !
   !   !   |   !7  !7 9|7 9!   !7 9
-----------------------------------
   !   !1 3|   !   !1  |1  !   !1 3
   ! 2 !   | 56! 4 ! 56| 56! 8 ! 56
7 9!   !7 9|7 9!   !7 9|7 9!   !7 9
-----------------------------------
   !1 3!1 3| 2 !12 !12 |12 !123!1 3
4  !4  !   | 56! 5 ! 56| 56! 56! 56
7 9!7 9!7 9|7 9!78 !789|7 9!7 9!7 9
===================================
 2 !1 3!123| 2 !123!12 |12 !123!1 3
45 !456!  6|456! 5 !456|456! 56!456
789!7 9!789|7 9!78 !789|789!7 9!7 9
-----------------------------------
 2 !1 3!123| 2 !123!12 |12 !123!1 3
45 !456!  6|456! 5 !456|456! 56!456
789!7 9!789|7 9!78 !789|789!7 9!7 9
-----------------------------------
 2 !1 3!123| 2 !123!12 |12 !123!1 3
45 !456!  6|456! 5 !456|456! 56!456
789!7 9!789|7 9!78 !789|789!7 9!7 9
```

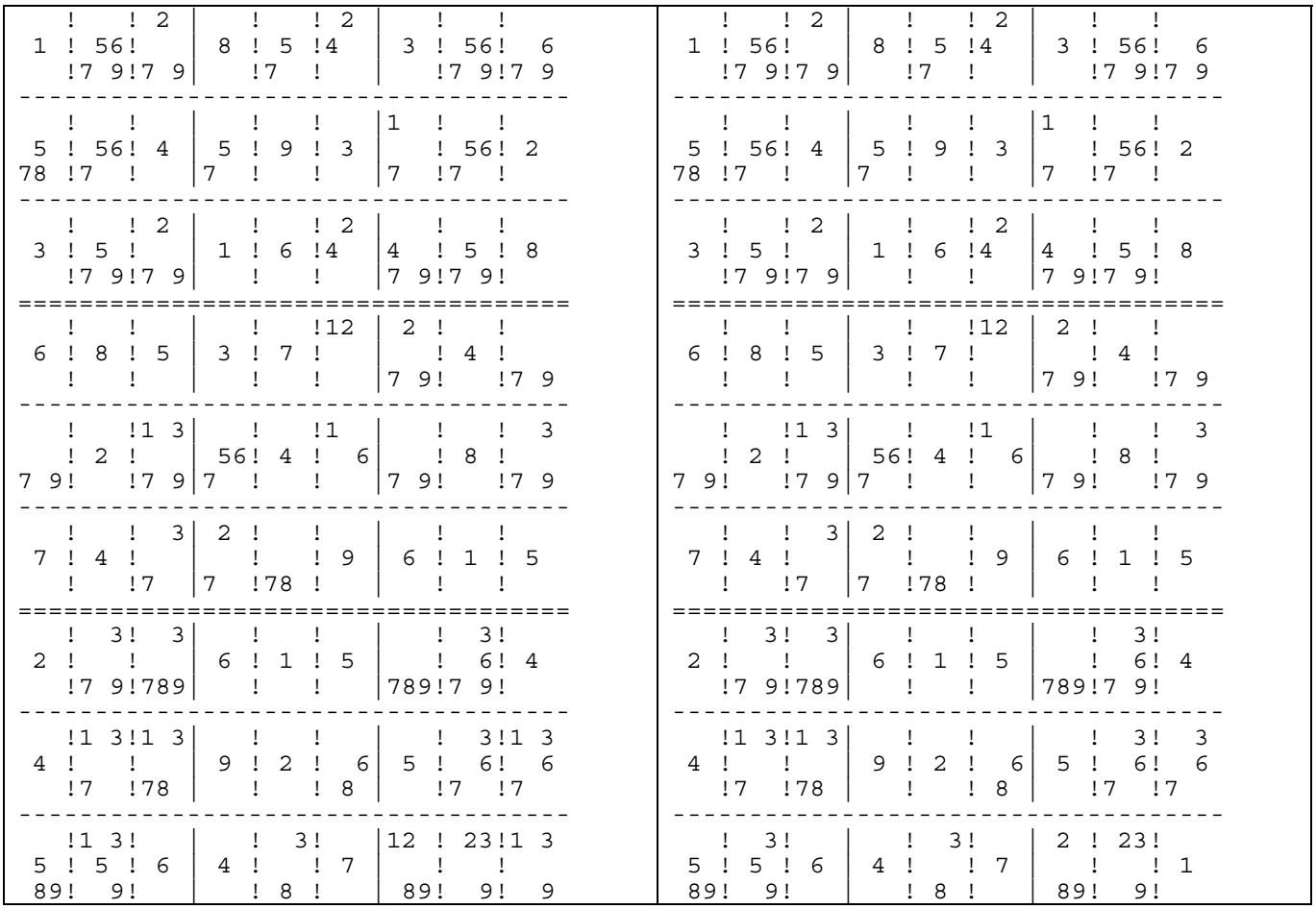
```
sudoku.updateSections(5, 1, 4);
sudoku.updateSections(5, 6, 6);
sudoku.updateSections(5, 8, 5);
sudoku.updateSections(6, 4, 1);
sudoku.updateSections(6, 8, 4);
```

```
sudoku.updateSections(5, 5, 9);
sudoku.updateSections(5, 7, 1);
sudoku.updateSections(6, 0, 2);
sudoku.updateSections(6, 5, 5);
sudoku.updateSections(7, 0, 4);
```



```
sudoku.updateSections(7, 3, 9);
sudoku.updateSections(7, 6, 5);
sudoku.updateSections(8, 5, 7);
```

```
sudoku.updateSections(7, 4, 2);
sudoku.updateSections(8, 2, 6);
sudoku.updateSections(8, 8, 1);
```



Solving the last step of the puzzle using the processing pairs method