CIT 149: Java I
Chapter 7 Lab1

In this lab we will complete programming project number 12 on page 566. For this chapter I will not include flowcharts. By this time you should understand the flow of the programs fairly well. This program uses multidimensional arrays to form a Sudoku puzzle.

1. Open a new document window in TextPad and save the program as SudokuPuzzle.java.
2. Type the appropriate block comments as in previous labs.
3. Type the code that will import the Scanner class.
4. Type the class header and opening brace. Within the class declare the following variables:

   ```
   private int board[][];
   private int start[][];
   ```

   - These are two multidimensional arrays. The brackets are for the number or rows and cols, respectively.

5. Type the following code that will create a constructor method:

   ```
   public SudokuPuzzle()
   {
      start = new int[9][9];
      board = new int[9][9];
   }
   ```

   - Here we set the initial rows and cols of the starting point of the puzzle, and the Sudoku board.

6. The toString() method will be used to set the string for the board. Type:

```
public String toString()
{
    String puzzleString = "Row/Col\n    1  2  3  4  5  6  7  8  9\n";
    puzzleString = puzzleString + "   -------------------------\n";
    for(int i=0; i<9; i++)
    {
        puzzleString = puzzleString + (i+1) + " |";
        for(int j=0; j<9; j++)
        {
            if(board[i][j] == 0)
                puzzleString = puzzleString +" "+ ".|";
            else
                puzzleString = puzzleString + " "+board[i][j] + "|";
        }
        puzzleString = puzzleString + "\n";
        puzzleString = puzzleString + "  |__|__|__|__|__|__|__|__|__|\n";
    }
    return puzzleString;
}
```

- **Note: There are four spaces before the number 1 and 2 spaces between each number. There are three spaces before the first hyphen and a total of 26 hyphens. On the line within the for loop there is a space before | (this is the pipe symbol above the \ on the keyboad). In the else statement there is a space between the two, double quotes. In the last line of the other for loop there are two spaces before the first | and there are two underscores (not hyphens) between each | symbol. Be cautious, watching out for spaces between some of the double quotes within this class. They are necessary when there is one.**
- Let's take a look at this carefully since it can be quite confusing. The values for the board are decided later on in the initializePuzzle() method which sends values to the addInitial() method. The toString()

method simply checks the values sent to that method since its responsible for the initial values in on the board. The initializePuzzle() method will receive the value of a new SudokuPuzzle object which is the same as declaring an new object.

- The toString() method first sets up the text "Row/Col", the next row, followed by a row of dashes. The outer for loop also sets up the first column for each row to have the value of i+1 and a |.
- The nested for loop handles the remainder of the columns in each row. The values for that row are actually given by the addInitial() method which maintains a record of the numbers. This for loop simply checks to see if the value is zero. If it is then a period is displayed; otherwise the value of the number is displayed.
- A new line is displayed and a series of underscores and pipe symbols are used to separate the different columns (the pipe symbol is located above the \ on most keyboards.

7. Our next method is the addInitial() method. Type:

```
public void addInitial(int row, int col, int value)
{
   if(row>=0 && row<=9 && col>=0 && col<=9 && value>=1 &&
      value<=9)
   {
      start[row][col] = value;
      board[row][col] = value;
   }
}
```

- This method receives values of three integers representing the row, column and the value to appear in that column in the row.
- An if statement states:
    - row must be greater or equal to 0 and less than or equal to 9 AND col must be the same, AND value must be greater or equal to 1 AND less than or equal to 9.

- if the above continues are true that the *start* and *board* multidimensional array will receive the value of the *value* variable passed to the method for the particular row and column. For example if the values passed to the method were 3, 4, 8 than the value of 8 would appear in row 3, column 4.

8. The addGuess() method is responsible for adding your guess to a particular column. Type:

```
public void addGuess(int row, int col, int value)
{
    // only set the value if the start is 0
   if(row>=0 && row<=9 && col>=0 && col<=9 && value>=1 &&
       value<=9 && start[row][col] == 0)
   {
       board[row][col] = value;
   }
}
```

- This is nearly identical to the previous method except there's an addition to the if statement, which is start[row][col] == 0. This means that the array value for this row and column must be zero. Recall that in the toString() method it placed a period where a *board* value was zero. We want to only be able to make a guess where there's a period.
- If the statement is true than the particular row and column will receive the value passed to the method.

9. The getValueIn() method will simply return the value in a particular row or column. Type:

```
public int getValueIn(int row, int col)
{
   return board[row][col];
}
```

10. The reset() method will set the *board* to the *start* array. Type:

```
public void reset()
{
   for(int i=0; i<9; i++)
       for(int j=0; j<9; j++)
```

```
            board[i][j] = start[i][j];
    }
```

- Notice that there are no braces for the for loops? This is because the outer for loop will be responsible for only the inner loop and no other code. The inner for loop is responsible for only a single line of code.

11. The isFull() method will check to see if all cols are full and return true or false. Type:

```
public boolean isFull()
{
    boolean allFilled = true;
    for(int i=0; i<9; i++)
        for(int j=0; j<9; j++)
            allFilled = allFilled && board[i][j]>0;
    return allFilled;
}
```

- Notice that the allFilled boolean checks to see if the variable is true AND whether the value on the board is greater than 0.

12. We also need to check for allowed values. The method of getAllowedValues() is responsible for this. Type:

```
public boolean[] getAllowedValues(int row, int col)
{
    // Save the value at the location, then try all 9 values
    int savedValue = board[row][col];
    boolean result[] = new boolean[9];
    for(int value = 1; value <=9; value++)
    {
        board[row][col] = value;
        result[value-1] = checkPuzzle();
    }
```

```
    board[row][col] = savedValue;
    return result;
}
```

- First an integer named *savedValue* receives the value of the board location. The location of the board has been passed to the method.
- A new boolean array is constructed and set to a limit of 9 elements. Recall that array elements have index numbers starting with zero.
- The for loop works as always. It has a starting point of 1 and an ending point of less than or equal to 9. The board location receives the value of the variable *value*, and the boolean array element at location of *value – 1* receives the value returned by the checkPuzzle() method.
- The board at the specified location receives the value of the variable *savedValue*.
- The value of the boolean *result* is returned where the method is invoked.

13. The checkPuzzle() method sees if the values in the square are legal or not. Type:

```
public boolean checkPuzzle()
{
    boolean looksGood = true;

    for(int i=0; i<9; i++)
    {
        looksGood = looksGood && checkRow(i);
        looksGood = looksGood && checkCol(i);
        looksGood = looksGood && checkSub(i);
    }
    return looksGood;
```

```
}
```

- Here a for loop is used with a starting point of 0 and ending point of less than 9. This is because the elements of the array will be between 0-9.
- The boolean looksGood receives its value from three different places.
    - checkRow() method which is passed the value of i.
    - checkCol() method which is also passed the value of i.
    - checkSub() method which is also passed the value of i.
    - We will look at each of these methods to see what they do.

14. The checkRow() method checks to make sure a number does not appear twice in a row. Type:

```
public boolean checkRow(int row)
{
   int count[] = new int[10];

   for(int col=0; col<9; col++)
   {
      count[board[row][col]]++;
   }

   boolean countIsOk = true;

   for(int i=1; i<=9; i++)
      countIsOk = countIsOk && (count[i]<=1);
   return countIsOk;
}
```

- An array is constructed and set to a limit of 10 elements. Although our puzzle has grids of 9, the first column numbers the rows.
- A for loop is much like we have been using all along. The array at element defined by the board location, has 1 added to it. This can be

rather confusing. Let's see an example. Say that the value of the variable *row*, passed to the method, is 3, and the value of *col* in the for loop is currently 3 as well. Let's make the assumption that the value at that point is 6. This means that the element in the count array with an index number of 6 will increase its value by 1.

- After the for loop a new boolean is constructed and set to true.
- A second for loop with a starting point of 1 and an ending point of 9 uses the boolean to check to see if the count is fine.
    - countIsOK's value is changed to true if its previous value was true AND the count array at element i is less than or equal to 1.
- Finally the value of countIsOk is returned.

15. The checkCol() method works the same as the checkRow(). Type:

```
public boolean checkCol(int col)
{
    int count[] = new int[10];

    for(int row=0; row<9; row++)
    {
        count[board[row][col]]++;
    }

    boolean countIsOk = true;

    for(int i=1; i<=9; i++)
        countIsOk = countIsOk && (count[i]<=1);

    return countIsOk;
}
```

16. The checkSub() method checks the 3 x 3 grid to make sure a number appears only once. Type:

```
public boolean checkSub(int sub)
{
    int count[] = new int[10];
    int rowBase = (sub/3) *3;
    // The above will give 0, 3, or 6 because of integer division
    int colBase = (sub%3) *3;

    for(int i=0; i<3; i++)
    {
        for(int j=0; j<3; j++)
        {
            count[board[rowBase+i][colBase+j]]++;
        }
    }
    boolean countIsOk = true;
    for(int i=1; i<=9; i++)
        countIsOk = countIsOk && (count[i]<=1);

    return countIsOk;
}
```

17. The initializePuzzle() method receives an argument of a new SudokuPuzzle object and passes values for each col for the object. Type:

```
public static void initializePuzzle(SudokuPuzzle p)
{

    p.addInitial(0,0,1);
    p.addInitial(0,1,2);
    p.addInitial(0,2,3);
    p.addInitial(0,3,4);
    p.addInitial(0,4,9);
    p.addInitial(0,5,7);
    p.addInitial(0,6,8);
    p.addInitial(0,7,6);
    p.addInitial(0,8,5);

    p.addInitial(1,0,4);
    p.addInitial(1,1,5);
    p.addInitial(1,2,9);

    p.addInitial(2,0,6);
    p.addInitial(2,1,7);
    p.addInitial(2,2,8);

    p.addInitial(3,0,3);
    p.addInitial(3,4,1);

    p.addInitial(4,0,2);

    p.addInitial(5,0,9);
    p.addInitial(5,5,5);

    p.addInitial(6,0,8);

    p.addInitial(7,0,7);
```

```
        p.addInitial(8,0,5);
        p.addInitial(8,3,9);
}
```

18. The final method is the main method. Type:

```
public static void main(String[] args)
{
    Scanner reader = new Scanner(System.in);

    System.out.println("Sudoku Game: ");

    SudokuPuzzle puzzle = new SudokuPuzzle();
    puzzle.initializePuzzle(puzzle);

    System.out.print("The puzzle is: \n" + puzzle);

    boolean done = false;
    while(!done)
    {
        System.out.println("What would you like to do? \n" +
          "Clear Puzzle(C) Set a square (S) Get possible values (G) Quit (Q)
           ");
        String response = reader.next();
        response = response.toLowerCase();

        if(response.equals("q"))
        {
            System.out.println("Thanks for playing.");
            done = true;
        }
        else if(response.equals("s"))
        {
```

```
    System.out.println("Which row (1-9) and colume (1-9) do you
      want to change?");
    int row = reader.nextInt()-1;
    int col = reader.nextInt()-1;
    System.out.println("What should the value (1-9) be?");
    int value = reader.nextInt();

    puzzle.addGuess(row, col, value);
}
else if(response.equals("g"))
{
    System.out.println("Which row (1-9) and colume (1-9) do you
      want to get values for?");
    int row = reader.nextInt()-1;
    int col = reader.nextInt()-1;
    boolean valid[] = puzzle.getAllowedValues(row, col);
    System.out.print("Allowed values are: ");

    for(int i=0; i<9; i++)
    {
        if(valid[i])
            System.out.print((i+1)+ "  ");
    }
    System.out.println();
}
else if(response.equals("c"))
{
    puzzle.reset();
}
System.out.print("The puzzle is now: \n" + puzzle);

if(!puzzle.checkPuzzle())
    System.out.println("You have made an error in the puzzle.");
else if(puzzle.isFull())
    System.out.println("Congratulations, you have completed the
```

```
            puzzle.");
    }
}
```

This is quite a long method, so let's check it out:

- A Scanner object is constructed
- Text is displayed.
- A new SudokuPuzzle object is constructed.
- The initializePuzzle() method is invoked for this object. The method is passed the value of the object.
- The initial puzzle is displayed.
- A boolean is constructed and set to false.
- while the boolean is not true:
    - The user is asked what they want to do and their response is assigned to the newly created variable *response*.
    - The value of the variable is set to lowercase.
    - if the user answered "q" then the while loop ends, after thanking the user for playing.
    - if the user answered "s" the user is asked to enter the row and column. Their response is assigned to two variables representing the rows and cols. A -1 is added because elements in arrays start with an index of zero.
    - the user is asked what value they want to place in the column in the specified row. Their entry is assigned to a variable named *value*.
    - the addGuess() method for the puzzle is invoked and passed the values of the *row*, *col*, and *value*.
    - if the user answered "g" they will be asked which row and column that want to get the value for.
    - A boolean array named valid receives the size returned by the getAllowedValues() method. The getAllowedValues() method is passed the values for *row* and *col* so it can perform its function.

        -
        - A for loop has a starting point of 0 and an ending point of less than 9. Within the loop:
            - an if statement is run if *valid* element *i* is true.
            - if the statement is true the number i + 1 is displayed.
        - if the user entered "c" the puzzle is reset by invoking the reset() method.
    - The puzzle is displayed with its update.
    - if the boolean returned by the checkPuzzle() method is not true then:
        - An error is displayed.
    - else if the boolean returned by the isFull() method is true:
        - A message of congratulations is displayed.

19. Close the class and compile the program. Fix any errors you may have. Test the program.
20. Compress the following files into a single zip or rar file and submit to the appropriate drop box:

    SudokuPuzzle.java
    SudokuPuzzle.class