Java应用技术homework1

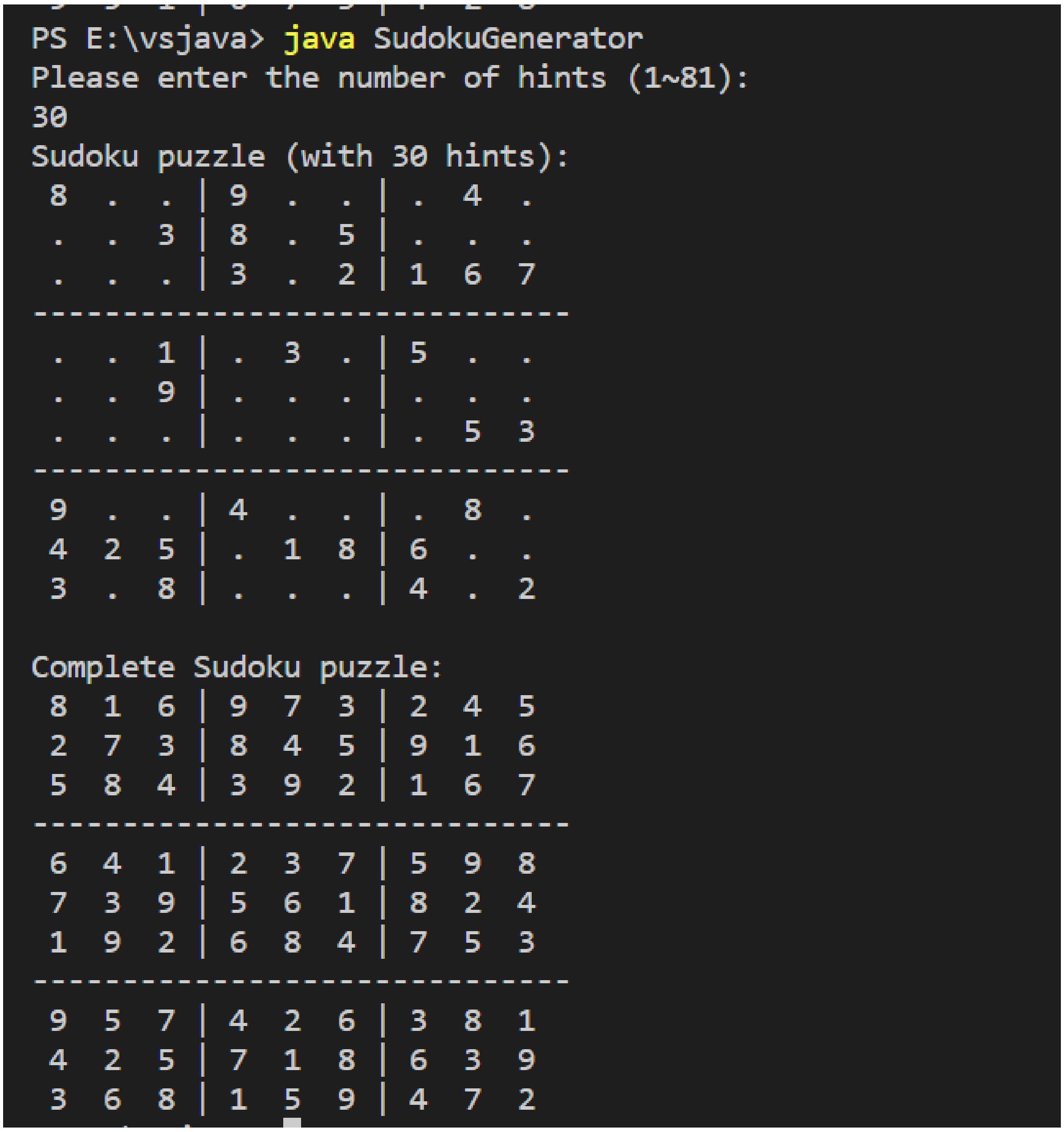
蔡佳伟 3220104519

一、数独代码

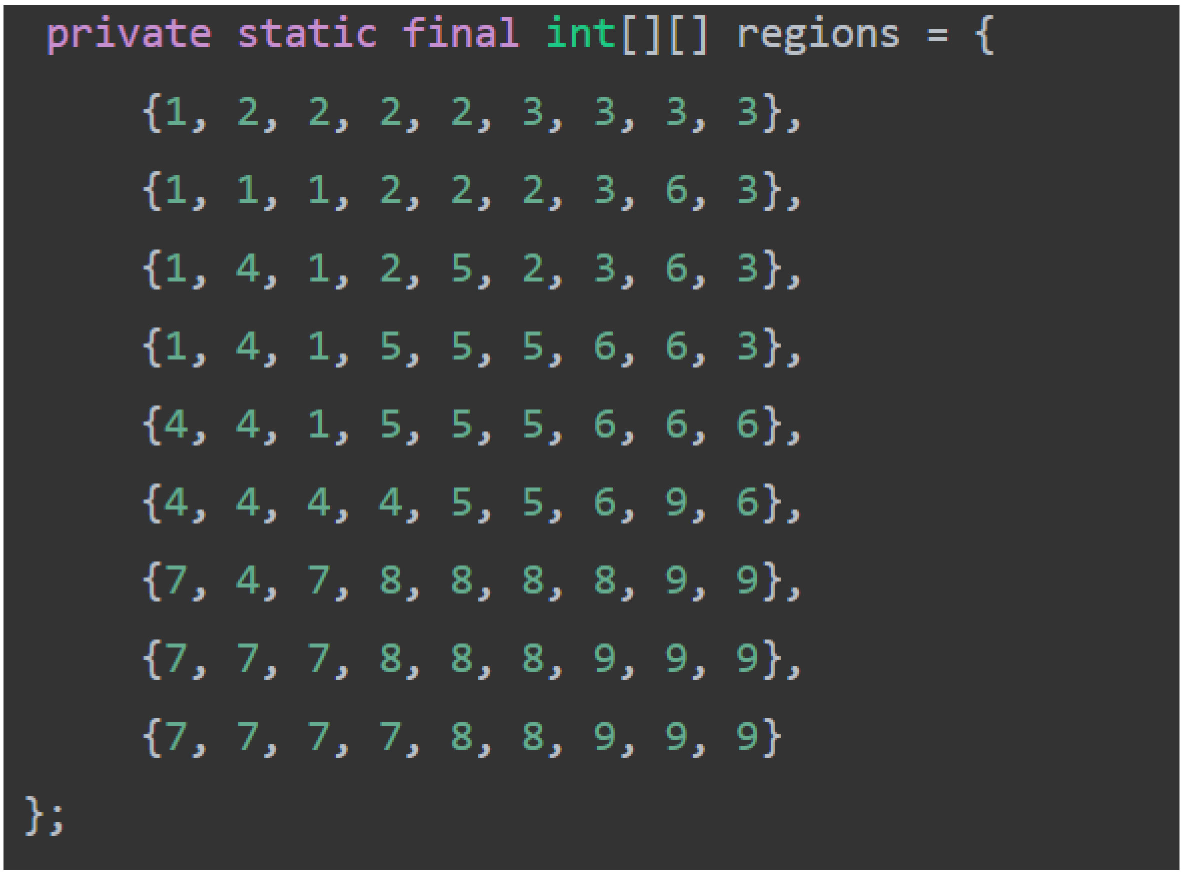
```java
import java.util.*;

public class SudokuGenerator {
    // Define the size of the Sudoku grid
    private static final int GRID_SIZE = 9;

    // Predefined custom regions where each number represents a distinct region
    private static final int[][] regions = {
        {1, 2, 2, 2, 2, 3, 3, 3, 3},
        {1, 1, 1, 2, 2, 2, 3, 6, 3},
        {1, 4, 1, 2, 5, 2, 3, 6, 3},
        {1, 4, 1, 5, 5, 5, 6, 6, 3},
        {4, 4, 1, 5, 5, 5, 6, 6, 6},
        {4, 4, 4, 4, 5, 5, 6, 9, 6},
        {7, 4, 7, 8, 8, 8, 8, 9, 9},
        {7, 7, 7, 8, 8, 8, 9, 9, 9},
        {7, 7, 7, 7, 8, 8, 9, 9, 9}
   };
    // private static final int[][] regions = {
    //     {1, 1, 1, 2, 2, 2, 3, 3, 3},
    //     {1, 1, 1, 2, 2, 2, 3, 3, 3},
    //     {1, 1, 1, 2, 2, 2, 3, 3, 3},
    //     {4, 4, 4, 5, 5, 5, 6, 6, 6},
    //     {4, 4, 4, 5, 5, 5, 6, 6, 6},
    //     {4, 4, 4, 5, 5, 5, 6, 6, 6},
    //     {7, 7, 7, 8, 8, 8, 9, 9, 9},
    //     {7, 7, 7, 8, 8, 8, 9, 9, 9},
    //     {7, 7, 7, 8, 8, 8, 9, 9, 9}
    // };


    // Main function
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input hint count (between 1 and 81) from the command line
        int hintCount = 0;
        while (hintCount < 1 || hintCount > 81) {
            System.out.println("Please enter the number of hints (1~81):");
            hintCount = scanner.nextInt();
        }

        // Generate a complete Sudoku board
        int[][] board = generateSudoku();
        // Generate a Sudoku puzzle based on the hint count
        int[][] puzzle = generatePuzzle(board, hintCount);

        // Print the generated Sudoku puzzle
        System.out.println("Sudoku puzzle (with " + hintCount + " hints):");
        printBoard(puzzle);

        // Print the complete Sudoku board
        System.out.println("\nComplete Sudoku puzzle:");
        printBoard(board);

    }

    // Generate a complete Sudoku puzzle
    private static int[][] generateSudoku() {
        int[][] board = new int[GRID_SIZE][GRID_SIZE];
        fillSudoku(board);
        return board;
    }

    // Fill the Sudoku board using a backtracking algorithm
    private static boolean fillSudoku(int[][] board) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);

        // Traverse each cell of the Sudoku board
        for (int row = 0; row < GRID_SIZE; row++) {
            for (int col = 0; col < GRID_SIZE; col++) {
                // If the current cell is empty
                if (board[row][col] == 0) {
                    Collections.shuffle(numbers);
                    for (int number : numbers) {
                        if (isValid(board, row, col, number)) {
                            board[row][col] = number;

                            // Recursively fill the next cell
                            if (fillSudoku(board)) {
                                return true;
                            }

                            // Backtrack if filling is not possible
                            board[row][col] = 0;
                        }
                    }
                    return false; // Backtrack
                }
            }
        }
        return true; // Successfully filled the board
    }

    // Check if placing a number is valid according to Sudoku rules
    private static boolean isValid(int[][] board, int row, int col, int num) {
        // Check the row and column
        for (int i = 0; i < GRID_SIZE; i++) {
            if (board[row][i] == num || board[i][col] == num) {
```

```java
                return false;
            }
        }

        // Check the custom region
        int region = regions[row][col];
        for (int i = 0; i < GRID_SIZE; i++) {
            for (int j = 0; j < GRID_SIZE; j++) {
                if (regions[i][j] == region && board[i][j] == num) {
                    return false;
                }
            }
        }

        return true;
    }

    // Generate a Sudoku puzzle with the specified number of hints
    private static int[][] generatePuzzle(int[][] board, int hintCount) {
        int[][] puzzle = new int[GRID_SIZE][GRID_SIZE];
        for (int i = 0; i < GRID_SIZE; i++) {
            puzzle[i] = Arrays.copyOf(board[i], GRID_SIZE);
        }

        // Create a list of all cell positions
        List<int[]> positions = new ArrayList<>();
        for (int row = 0; row < GRID_SIZE; row++) {
            for (int col = 0; col < GRID_SIZE; col++) {
                positions.add(new int[]{row, col});
            }
        }

        // Shuffle the positions randomly
        Collections.shuffle(positions);

        // Hide (mask) 81 - hintCount cells
        int cellsToMask = GRID_SIZE * GRID_SIZE - hintCount;
        for (int i = 0; i < cellsToMask; i++) {
            int[] pos = positions.get(i);
            puzzle[pos[0]][pos[1]] = 0; // Set the cell value to 0 (empty)
        }

        return puzzle;
    }

    // Print the Sudoku board
    private static void printBoard(int[][] board) {
        for (int row = 0; row < GRID_SIZE; row++) {
            if (row % 3 == 0 && row != 0) {
                System.out.println("------------------------------");
            }
            for (int col = 0; col < GRID_SIZE; col++) {
                if (col % 3 == 0 && col != 0) {
                    System.out.print("|");
                }
                if (board[row][col] == 0) {
                    System.out.print(" . ");
                } else {
                    System.out.print(" " + board[row][col] + " ");
                }
            }
            System.out.println();
        }
    }
}
```

二、代码说明和结果



```
PS E:\vsjava> java SudokuGenerator
Please enter the number of hints (1~81):
30
Sudoku puzzle (with 30 hints):
 8  .  . | 9  .  . | .  4  .
 .  .  3 | 8  .  5 | .  .  .
 .  .  . | 3  .  2 | 1  6  7
------------------------------
 .  .  1 | .  3  . | 5  .  .
 .  .  9 | .  .  . | .  .  .
 .  .  . | .  .  . | .  5  3
------------------------------
 9  .  . | 4  .  . | .  8  .
 4  2  5 | .  1  8 | 6  .  .
 3  .  8 | .  .  . | 4  .  2

Complete Sudoku puzzle:
 8  1  6 | 9  7  3 | 2  4  5
 2  7  3 | 8  4  5 | 9  1  6
 5  8  4 | 3  9  2 | 1  6  7
------------------------------
 6  4  1 | 2  3  7 | 5  9  8
 7  3  9 | 5  6  1 | 8  2  4
 1  9  2 | 6  8  4 | 7  5  3
------------------------------
 9  5  7 | 4  2  6 | 3  8  1
 4  2  5 | 7  1  8 | 6  3  9
 3  6  8 | 1  5  9 | 4  7  2
```



```
private static final int[][] regions = {
    {1, 2, 2, 2, 2, 3, 3, 3, 3},
    {1, 1, 1, 2, 2, 2, 3, 6, 3},
    {1, 4, 1, 2, 5, 2, 3, 6, 3},
    {1, 4, 1, 5, 5, 5, 6, 6, 3},
    {4, 4, 1, 5, 5, 5, 6, 6, 6},
    {4, 4, 4, 4, 5, 5, 6, 9, 6},
    {7, 4, 7, 8, 8, 8, 8, 9, 9},
    {7, 7, 7, 8, 8, 8, 9, 9, 9},
    {7, 7, 7, 7, 8, 8, 9, 9, 9}
};
```

我是在vscode下运行，使用`javac SudokuGenerator.java`编译出`SudokuGenerator.class`文件，然后使用`java SudokuGenerator`即可运行，运行后先输入提示数的个数（这里我输入30），然后会给出数独谜题提示（含有30个数），然后会输出一种解答。其中可以看到每一个region内的9个数都是不一样的（可以经过如上对照）。

三、代码思路解释

3.1 region划分

```
private static final int[][] regions = {
        {1, 2, 2, 2, 2, 3, 3, 3, 3},
        {1, 1, 1, 2, 2, 2, 3, 6, 3},
        {1, 4, 1, 2, 5, 2, 3, 6, 3},
        {1, 4, 1, 5, 5, 5, 6, 6, 3},
        {4, 4, 1, 5, 5, 5, 6, 6, 6},
        {4, 4, 4, 4, 5, 5, 6, 9, 6},
        {7, 4, 7, 8, 8, 8, 8, 9, 9},
        {7, 7, 7, 8, 8, 8, 9, 9, 9},
        {7, 7, 7, 7, 8, 8, 9, 9, 9}
  };
   // private static final int[][] regions = {
   //     {1, 1, 1, 2, 2, 2, 3, 3, 3},
   //     {1, 1, 1, 2, 2, 2, 3, 3, 3},
   //     {1, 1, 1, 2, 2, 2, 3, 3, 3},
   //     {4, 4, 4, 5, 5, 5, 6, 6, 6},
   //     {4, 4, 4, 5, 5, 5, 6, 6, 6},
   //     {4, 4, 4, 5, 5, 5, 6, 6, 6},
   //     {7, 7, 7, 8, 8, 8, 9, 9, 9},
   //     {7, 7, 7, 8, 8, 8, 9, 9, 9},
   //     {7, 7, 7, 8, 8, 8, 9, 9, 9}
   // };
```

在这里，我手动把9*9的方格划分成了9个区域，每个区域有9个位置，之后的数独在每个区域的数字都不会重复。

3.2 主函数部分

```
 // Main function
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input hint count (between 1 and 81) from the command line
        int hintCount = 0;
        while (hintCount < 1 || hintCount > 81) {
            System.out.println("Please enter the number of hints (1~81):");
            hintCount = scanner.nextInt();
        }

        // Generate a complete Sudoku board
        int[][] board = generateSudoku();

        // Generate a Sudoku puzzle based on the hint count
        int[][] puzzle = generatePuzzle(board, hintCount);

        // Print the generated Sudoku puzzle
        System.out.println("Sudoku puzzle (with " + hintCount + " hints):");
        printBoard(puzzle);

        // Print the complete Sudoku board
        System.out.println("\nComplete Sudoku puzzle:");
        printBoard(board);

    }
```

主函数负责调用各个部分的函数，从键盘中输入获取提示数，然后调用函数生成谜题和解答，最后打印数独。

3.3 生成数独

```
// Generate a complete Sudoku puzzle
    private static int[][] generateSudoku() {
        int[][] board = new int[GRID_SIZE][GRID_SIZE];
        fillSudoku(board);
        return board;
    }

    // Fill the Sudoku board using a backtracking algorithm
    private static boolean fillSudoku(int[][] board) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);

        // Traverse each cell of the Sudoku board
        for (int row = 0; row < GRID_SIZE; row++) {
            for (int col = 0; col < GRID_SIZE; col++) {
                // If the current cell is empty
                if (board[row][col] == 0) {
```

```java
                    Collections.shuffle(numbers);
                    for (int number : numbers) {
                        if (isValid(board, row, col, number)) {
                            board[row][col] = number;

                            // Recursively fill the next cell
                            if (fillSudoku(board)) {
                                return true;
                            }

                            // Backtrack if filling is not possible
                            board[row][col] = 0;
                        }
                    }
                    return false; // Backtrack
                }
            }
        }
        return true; // Successfully filled the board
    }
  // Check if placing a number is valid according to Sudoku rules
    private static boolean isValid(int[][] board, int row, int col, int num) {
        // Check the row and column
        for (int i = 0; i < GRID_SIZE; i++) {
            if (board[row][i] == num || board[i][col] == num) {
                return false;
            }
        }

        // Check the custom region
        int region = regions[row][col];
        for (int i = 0; i < GRID_SIZE; i++) {
            for (int j = 0; j < GRID_SIZE; j++) {
                if (regions[i][j] == region && board[i][j] == num) {
                    return false;
                }
            }
        }

        return true;
    }
```

`generateSudoku`函数负责生成完整数独谜题，其中`fillSudoku`函数使用dfs递归回溯算法来实现。每次向方格填入数字时，会先随机填入，然后继续尝试填入下一个位置。如果下一个位置无法填入，则进行回溯。

其中判断能否填入，先遍历一整个横行和一整个纵列，判断不能有重复，再遍历整张图，找到region数组值相同的位置，这些位置按照mask划分为同一个区域，也不能有数字重复。如果有重复，则返回false，告知不能填入，否则填入true，告知可以填入。

3.4 生成谜题（挖去空格）

```java
// Generate a Sudoku puzzle with the specified number of hints
    private static int[][] generatePuzzle(int[][] board, int hintCount) {
        int[][] puzzle = new int[GRID_SIZE][GRID_SIZE];
        for (int i = 0; i < GRID_SIZE; i++) {
            puzzle[i] = Arrays.copyOf(board[i], GRID_SIZE);
        }

        // Create a list of all cell positions
        List<int[]> positions = new ArrayList<>();
        for (int row = 0; row < GRID_SIZE; row++) {
            for (int col = 0; col < GRID_SIZE; col++) {
                positions.add(new int[]{row, col});
            }
        }

        // Shuffle the positions randomly
        Collections.shuffle(positions);

        // Hide (mask) 81 - hintCount cells
        int cellsToMask = GRID_SIZE * GRID_SIZE - hintCount;
        for (int i = 0; i < cellsToMask; i++) {
            int[] pos = positions.get(i);
            puzzle[pos[0]][pos[1]] = 0; // Set the cell value to 0 (empty)
        }

        return puzzle;
    }
```

生成谜题部分，我们先复制了数组完整填完的结果，这样操作是为了生成的题目肯定有解。如果直接随意生成数组，很有可能会生成一道无解的题目，造成程序死循环等。收到键盘输入的提示数个数后，我们首先复制了一个完整的解答到puzzle数组中，然后对puzzle数组随机打乱，然后对随机后的puzzle数组的一部分赋值为0，这样可以确保生成的结果比较平均，每行、每列、每格中空格能尽可能均匀分布。

四、心得体会

这次java数独题目设计本身不算困难，本身算法在前两年中我已经比较熟悉了。这次实验主要让我学会了很多java函数的使用知识，例如 `shuffle()`，`Arrays.copyOf()` 等函数。我也进行了一定的封装，使我对java面向对象的技术特点有了很好的认识。

在过程中，我也遇到了一些困难，比如dfs实现错误，不知道mask怎么划分（现在其实我也不确定，就直接在代码中手动进行了划分），不知道怎么实现一定有解的谜题，不知道怎么清除数据使得尽可能均匀分布等，不过最后这些问题都得以解决。这次作业也培养了我解决困难的能力~