Java应用技术 Homework

基本信息

- 个人信息：徐文皓，3210102377
- 课程班级：2023-2024学年秋冬学期，鲁伟明，周三9,10节
- 作业日期：2023.10.21

作业要求

- 编写Sudoku生成器
  - 根据用户在命令行输入的提示数，自动生成Sudoku题目，要求每行、每列、每格中空格能尽可能均匀分布
- 编写Sudoku求解器
  - 从终端输入一个Sudoku题目，给出解答

作业内容

程序效果

进入欢迎界面，按照提示选择模式，可选模式有：生成器(1)、求解器(2)、退出程序(3)。

```
[Welcome] Welcome to the Sudoku Game.

[Select] Please select a mode among:
[Select] 1 - Problem Generator
[Select] 2 - Problem Solver
[Select] 3 - Quit
[Select] Your Choice:
```

选择Sudoku生成器，终端提示用户输入提示数的个数。为了保证能够生成有唯一解的题目，根据查阅到的资料以及反复实验，提示数应当不低于22。在此时输入0可以返回欢迎界面。

我们输入25，可以观察到终端输出了一道有25个提示数的Sudoku题目，并提示用户再次按下Enter键以查看答案。

```
[Note] To efficiently generate problems with unique solutions,
[Note] the numbers initially provided must be no less than 22.
[Note] Input 0 to quit the generator.
[Generator] Input the numbers initially provided: 25
[Init] Initializing the Sudoku with 25 numbers provided.

Sudoku:
\C 1 2 3 4 5 6 7 8 9
R\ -----------------
1| 2 _ 4 _ 3 _ _ _ _
2| 1 _ _ 2 _ _ _ _ 9
3| _ _ 7 _ 8 _ _ _ 5
4| 7 1 _ _ _ 4 8 _ _
5| _ _ _ 5 _ _ _ 1 3
6| _ _ 9 _ 7 _ _ _ _
7| _ _ 2 _ 1 _ _ _ _
8| _ _ _ 9 _ _ _ 8 _
9| _ _ _ _ 5 _ 4 3 _

[Note] Press Enter again to view the answer.
```

再次按下Enter键，终端输出该题目答案，并返回欢迎界面。

```
Solution:
\C 1 2 3 4 5 6 7 8 9
R\ ----------------
1| 2 9 4 1 3 5 6 7 8
2| 1 5 8 2 6 7 3 4 9
3| 3 6 7 4 8 9 1 2 5
4| 7 1 5 3 2 4 8 9 6
5| 4 2 6 5 9 8 7 1 3
6| 8 3 9 6 7 1 2 5 4
7| 5 4 2 8 1 3 9 6 7
8| 6 7 3 9 4 2 5 8 1
9| 9 8 1 7 5 6 4 3 2


[Select] Please select a mode among:
[Select] 1 - Problem Generator
[Select] 2 - Problem Solver
[Select] 3 - Quit
[Select] Your Choice:
```

可以发现，在生成器模式输入的内容不合法时，将给出对应的错误提示。

```
[Note] To efficiently generate problems with unique solutions,
[Note] the numbers initially provided must be no less than 22.
[Note] Input 0 to quit the generator.
[Generator] Input the numbers initially provided: q
[Error] Not a number.

[Note] To efficiently generate problems with unique solutions,
[Note] the numbers initially provided must be no less than 22.
[Note] Input 0 to quit the generator.
[Generator] Input the numbers initially provided: 15
[Error] The input should be between 22 and 81.
```

接下来进入Sudoku求解器模式。

终端提示用户输入题目，空位需要以`-`、`_`、`.`、`0`中的任何一种形式给出。如果输入题目的解不唯一，求解器将给出一个有效解。

```
[Note] Replace the blank with one of {'-','_','.','0'}.
[Note] If the solution is not unique,
[Note] only one legal solution will be presented.
[Solver] Input the problem:
```

我们以作业要求中的例题为例：

- Sudoku求解器
  - 输入一个数独题目（可终端输入）

```
8 . . . . . . . .
. . 3 6 . . . . .
. 7 . . 9 . 2 . .
. 5 . . . 7 . . .
. . . . 4 5 7 . .
. . . 1 . . . 3 .
. . 1 . . . . 6 8
. . 8 5 . . . 1 .
. 9 . . . . 4 . .
```

  - 系统自动给出解答，并从终端输出

```
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2
```

输入

```
8 . . . . . . . .
. . 3 6 . . . . .
. 7 . . 9 . 2 . .
. 5 . . . 7 . . .
. . . . 4 5 7 . .
. . . 1 . . . 3 .
. . 1 . . . . 6 8
. . 8 5 . . . 1 .
. 9 . . . . 4 . .
```

可以发现求解器给出了与作业要求中一致的答案。

```
[Solver] Input the problem:
8 . . . . . . . .
. . 3 6 . . . . .
. 7 . . 9 . 2 . .
. 5 . . . 7 . . .
. . . . 4 5 7 . .
. . . 1 . . . 3 .
. . 1 . . . . 6 8
. . 8 5 . . . 1 .
. 9 . . . . 4 . .

[Solver] Here is the solution:

Solution:
\C 1 2 3 4 5 6 7 8 9
R\ ----------------
1| 8 1 2 7 5 3 6 4 9
2| 9 4 3 6 8 2 1 7 5
3| 6 7 5 4 9 1 2 8 3
4| 1 5 4 2 3 7 8 9 6
5| 3 6 9 8 4 5 7 2 1
6| 2 8 7 1 6 9 5 3 4
7| 5 2 1 9 7 4 3 6 8
8| 4 3 8 5 2 6 9 1 7
9| 7 9 6 3 1 8 4 5 2
```

若无解，求解器将给出无解提示。

```
[Solver] Input the problem:
1 . . . . . . . .
. . 3 6 . . . . .
. 7 . . 9 . 2 . .
. 5 . . . 7 . . .
. . . . 4 5 7 . .
. . . 1 . . . 3 .
. . 1 . . . . 6 8
. . 8 5 . . . 1 .
. 9 . . . . 4 . .

[Solver] Here is the solution:
[Failed] No Solution.
```

具体实现

我们首先给出所有类及其属性、方法。

```java
public class Main {

    static Scanner in = new Scanner(System.in);

    public static void main(String args[]);
    static void display(); // 展示欢迎界面
    static void displayGenerator(Sudoku sudoku); // 引导用户使用Sudoku生成器
    static void displaySolver(Sudoku sudoku); // 引导用户使用Sudoku求解器

}

class Sudoku {

    int ans[][]; // 储存固定的提示数
    int sdk[][]; // 当前Sudoku状态
    int[][] removeSeq = new int[9][]; // 用于记录随机生成的每个宫的移除顺序
    int[] blockRemoved = { 0, 0, 0, 0, 0, 0, 0, 0, 0 }; // 用于记录每个宫已经移除的格子数

    Sudoku(); // 构造函数，用于对ans和sdk分配空间并初始化

    // 以下为通用的辅助方法
    boolean check(int pos); // 用于检查sdk的pos位置的数字是否与其他数字冲突
    void printSudoku(); // 用于打印当前sdk的状态
    int[] randomArray(int offset); // 用于生成一个有9个连续整数的序列
    void clear(); // 清除题目，用于将Sudoku恢复为初始状态
    void reset(); // 用于将Sudoku恢复为未作答状态，即使得sdk与ans相同

    // 以下为Sudoku生成器相关方法
```

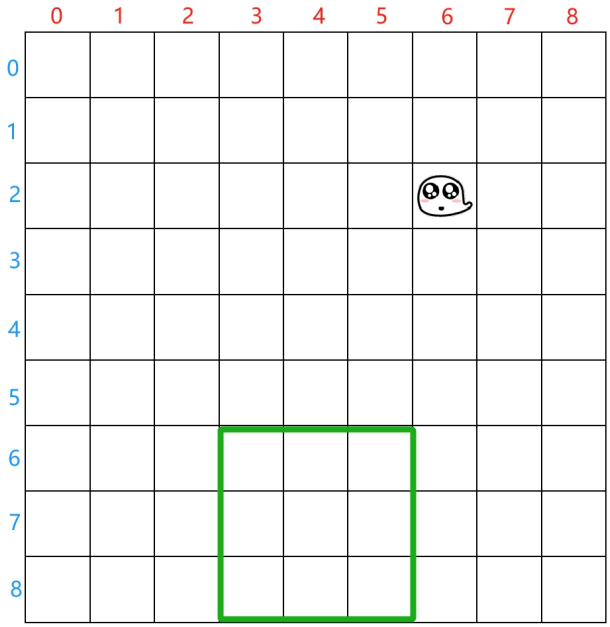
```
    void randomInit(int numOfQues); // 用于生成的核心函数
    void randomSet(); // 用于随机生成一个有效的9*9状态
    void resetRandomRemoveVar(); // 用于重置removeSeq和blockRemoved
    boolean randomRemove(int block); // 用于从指定宫中移除一个数字

    // 以下为Sudoku求解器相关方法
    boolean input(Scanner in); // 用于读取用户输入
    boolean backtracking(int[][] oper, int pos); // 用于进行回溯搜索
    boolean solve(boolean toModify, boolean toPrint); // 用于求解的核心函数

}
```

其中`Main`类的作用主要是组织`Sudoku`类的相关功能，并提供给用户使用接口，在这里我们不作赘述，源代码将展示在附录中。

在介绍`Sudoku`类的方法之前，我们先对`[i][j]`、`pos`和`block`这三个概念做出定义。、



Sudoku为一个9*9的数字矩阵，我们使用0~8对它的行和列进行编号。

`[i][j]`和`pos`形式均表示矩阵中的一个元素。对于`[i][j]`表示形式，`[i]`表示行号(Row，即蓝色数字标识)，`[j]`表示列号(Col，即红色数字标识)，因此`[i][j]`表示第`i`行的第`j`个元素；而对于`pos`表示形式，可以理解为按照从上至下、从左至右的顺序将元素进行编号，第一行分别为0,1,2,...,8，而第二行分别为9,10,11,...,17，以此类推。举例来说，上图中的表情所在的元素可以用`[2][6]`或者`24`表示。它们之间的对应关系为：

$$i = pos/9, \quad j = pos\%9$$

而`block`是指其中的一个“宫”，代表一个3*3的子矩阵。我们从上至下、从左至右将宫分别编号为0~8。如上图中绿色方框为第`7`宫。`block`和`[i][j]`之间的对应关系为：

$$block/3 \times 3 \leq i \leq block/3 \times 3 + 2, \quad block\%3 \times 3 \leq j \leq block\%3 \times 3 + 2$$

以下就每个方法给出具体说明：

通用辅助方法

1. `boolean check(int pos)`

   检查 pos 元素是否合法，合法则返回true，否则(与其他元素冲突)返回false。

```
boolean check(int pos) {
    将pos解析为[i][j]形式
    if 第i行的某个其他元素与sdk[i][j]相同 return false
    if 第j列的某个其他元素与sdk[i][j]相同 return false
    将[i][j]解析为block形式
    if 该block中的某个其他元素与sdk[i][j]相同 return false
    return true
}
```

2. `void printSudoku()`

   将Sudoku当前状态输出到终端。

```
void printSudoku() {
    打印表头信息
    for sdk中的每个元素item:
        if item > 0 打印该元素
        else 打印一个下划线'_'
}
```

3. `int[] randomArray(int offset)`

   生成一个含有9个连续整数、但位置随机的数组，其中offset为第一个整数的值。

   如randomArray(0)将生成0,1,...,8的随机重排结果，而randomArray(1)将生成1,2,..,9的随机重排结果。

```
int[] randomArray(int offset) {
    int[] result = new int[9]
    result中的数据分别为0+offset, 1+offset, ... , 8+offset
    对result数组进行随机重排
    return result.clone()
}
```

4. `void clear()`

   重置Sudoku。

```
void clear() {
    for Sudoku的每个item:
        sdk[i][j] = ans[i][j] = -1
}
```

5. `void reset()`

   将Sudoku恢复为未作答状态。

```
void reset() {
    for Sudoku的每个item:
        sdk[i][j] = ans[i][j]
}
```

Sudoku生成器

在这里，我们首先介绍`removeSeq`和`blockRemoved`的作用。

我们进行生成题目的策略，是先对部分元素进行随机确定，再调用求解器对这个Sudoku状态进行求解，得到一个有效解。之后，我们按照一定的顺序删除元素(“挖空”)，直到剩余元素数量符合用户要求。

为了保证删除的结果较为均匀，我们首先生成一个随机序列`removeBlockSeq`，是一个随机的0~8重排序列。按照`removeBlockSeq`的顺序，我们依次从对应的宫中删除元素。`removeSeq`是一个随机矩阵，其中的一个元素`removeSeq[block]`也是一个随机的0~8重排序列，这个序列的顺序是对应宫中9个元素的删除顺序。

这样，我们只要保证随机依序选择待删除的宫，并且在每个宫中随机依序删除元素，那么得到的结果就大概率是比较均匀的。

为了保证最终生成的题目有唯一解，在每次删除元素时我们都要检验当前状态是否符合要求(具体策略将在randomRemove方法中介绍)，如果该元素的删除会导致题目出现多解，那么我们在该宫中选择下一个元素删除；如果该宫中所有元素都不能被删除，那么我们就在下一个宫中尝试删除。`blockRemoved`记录了每个宫已经尝试删除过的元素数量，当`blockRemoved[block]==9`时，说明该宫中所有元素都已被尝试删除过，我们将`removeBlockSeq`中值为`block`的元素置为`-1`，表示这个宫不再接受删除请求。

在一次生成尝试中，我们需要保证`removeBlockSeq`、`removeSeq`、`blockRemoved`只被初始化(或随机化)一次。

1. `void randomInit(int numOfQues)`

   Sudoku生成器的核心方法。

```
void randomInit(int numOfQues) {
    打印开始生成的提示
    while true:
        调用resetRandomRemoveVar()重置随机序列
        调用randomSet()生成随机有效解
        int[] removeBlockSeq = randomArray(0)
        boolean successFlag = true
        for i from 0 to 80 - numOfQues: // 需要从有效解中移除81 - numOfQues个元素
            for tryRemove from 0 to 8: // 对9个宫按随机序列的顺序尝试移除元素
                if removeSeq[(i + tryRemove) % 9] == -1: // 如果该宫元素已经都不再可以被移除
                    continue
                if 调用randomRemove(removeSeq[(i + tryRemove) % 9]) == true: // 移除成功
                    break
                else: // 移除失败，说明这个宫已经不再可以被移除，将removeSeq对应元素置为-1
                    removeSeq[(i + tryRemove) % 9] = -1;
            if tryRemove == 9:  // 所有宫都不再可以被移除
                successFlag = false
```

```
        if successFlag: // 成功移除了81 - numOfQues个元素
            调用reset()将当前状态设置为与题目一致 // 因为求解时可能破坏了sdk的内容，在此将其复原
            break
        // 如果程序运行到这里，说明移除失败了，我们重新随机生成一次
        调用clear()将Sudoku清空
}
```

2. `void randomSet()`

我们首先设置好15个元素，之后对当前状态进行求解，得到一张完整的有效解。

> 在这里，我起初试图只设置好一列元素进行求解，但那样结合回溯搜索会出现答案中一行为(1,2,3,4,5,6,7,8,9)之类的情况。为此，经过实验，将一列和一宫设置好后再去求解可以很大程度上避免这种情况，增加了题目的随机性。

```
void randomSet() {
    int[] firstCol = randomArray(1)
    将第0列元素依次设置为firstCol的元素
    int[] firstBlock = randomArray(1)
    将第0宫元素依次设置为randomArray中与第0,9,18号元素不一致的元素 // 这一定不会导致冲突
    调用solve(true, false)求解
    for Sudoku中的每个item:
        ans[i][j] = sdk[i][j] // 将答案设置为求解结果
}
```

3. `void resetRandomRemoveVar()`

为保证随机性，在每次生成尝试前，我们重置`removeSeq`和`blockRemoved`。而`removeBlockSeq`将在`void randomInit(int numOfQues)`被自然重置。

```
void resetRandomRemoveVar() {
    for i from 0 to 8:
        removeSeq[i] = randomArray(0)
        blockRemoved[i] = 0
}
```

4. `boolean randomRemove(int block)`

在`block`宫中尝试删除一个元素，成功即返回true，否则(宫中所有元素都不接受被删除)返回false。

我们每删除一个元素时，需要验证删除这个元素是否会导致题目多解。验证的策略是，将这个元素中原来的数字替换成其他8个数字，如果某个元素可以求解，说明这个元素的删除会导致多解，从而我们拒绝这个元素的删除请求，转而向同宫的下一个元素发起删除请求。频繁的调用solve()会严重降低程序性能，因此我们先通过check()进行验证。

```
boolean randomRemove(int block) {
    while true:
        if blockRemoved[block] == 9: // 所有元素都不接受被删除
            return false
        从removeSeq[block]中定位即将被请求删除的元素[i][j]
        blockRemoved[block]++
        int temp = ans[i][j] // 存储ans[i][j]的值
        for ans[i][j] from 1 to 9: // 对这个元素是每个数字的情况
```

```
            if temp == ans[i][j]:
                continue
            调用reset()恢复Sudoku状态
            // 接下来尝试ans[i][j]是这种可能时是否会有解
            if check(i * 9 + j) == false: // 冲突，无解
                continue
            else if this.solve(false, false) == false: // 求解结果为无解
                continue
            else // 说明这个元素会导致多解
                break
        if  ans[i][j] == 10: // 说明该元素是每个数字的可能都已被考虑过，均不会导致有解，说明该元素被删除后仍有唯一解
            ans[i][j] = sdk[i][j] = 0 // 删除该元素
            return true;
        else: // 该元素的删除会导致多解
            ans[i][j] = sdk[i][j] = temp // 恢复至删除前状态
}
```

Sudoku求解器

1. `boolean input(Scanner in)`

   接收终端的题目输入。

```
boolean input(Scanner in) {

        boolean successFlag = true;
        String s;
        // Input
        for (int i = 0; i < 9; i++) {
            s = in.nextLine();
            if (s == "\n") {
                i--;
            }
            int index = 0;
            int j = 0;
            while (index < s.length()) {
                if (j == 9)
                    break;
                if (s.charAt(index) == '-' || s.charAt(index) == '_' || s.charAt(index) == '.') {
                    ans[i][j] = sdk[i][j] = 0;
                    j++;
                } else if (s.charAt(index) >= '0' && s.charAt(index) <= '9') {
                    ans[i][j] = sdk[i][j] = s.charAt(index) - '0';
                    j++;
                }
                index++;
            }
            if (j != 9)
                break;
        }
        // Check whether the input is valid
```

```
        for (int i = 0; i < 9; i++)
            for (int j = 0; j < 9; j++) {
                if (sdk[i][j] == -1)
                    successFlag = false;
            }

        return successFlag;
    }
```

2. `boolean backtracking(int[][] oper, int pos)`

   用于进行回溯求解，使用深度优先搜索方法，在发生冲突时剪枝。

```
boolean backtracking(int[][] oper, int pos) {

    boolean found = false

    if pos == 81: // 搜索结束，已经将矩阵填满
        return true

    将pos解析为[i][j]形式
    if ans[i][j] > 0: // 这个元素是题目给出的固定数字
        oper[i][j] = ans[i][j]
        found = backtracking(oper, pos + 1) // 跳过本元素，直接递归搜索下一个元素
    else: // 这个元素是要填的空
        for num from 1 to 9:
            oper[i][j] = num // 从1到9尝试
            if check(pos): // 满足条件，没有产生冲突
                found = backtracking(oper, pos + 1) // 递归搜索下一个元素
                if !found: // 递归返回，搜索失败
                    oper[i][j] = 0 // 将该元素重置
            else:
                oper[i][j] = 0 // 产生冲突，将该元素重置(剪枝)
            if found: // 递归返回，搜索成功
                break
    return found
}

```

3. `boolean solve(boolean toModify, boolean toPrint)`

   Sudoku求解器的核心方法。用于对当前sdk状态进行求解。其中涉及两个参数：

   - `boolean toModify`：设置为true时，将求解结果存储在sdk处；设置为false时，只判断当前sdk是否有解，不做修改。
   - `boolean toPrint`：设置为true时，打印求解结果；设置为false时，不打印求解结果，只通过返回值说明是否有解。

```
boolean solve(boolean toModify, boolean toPrint) {
    int[][] oper
    if toModify: // 要求修改sdk
        oper = sdk // 在sdk上操作
    else: // 要求不修改sdk
        oper = sdk.clone() // 在sdk的副本上操作
    boolean solved = backtracking(oper, 0) // 调用backtracking()求解
```

```
    if toPrint:
        if solved: // 有解
            打印题解
        else: // 无解
            打印无解提示
    return solved
}
```

源代码

```
// Main.java

import java.util.Scanner;

public class Main {

    static Scanner in = new Scanner(System.in);

    public static void main(String args[]) {
        display();
    }

    static void display() {
        Sudoku sudoku = new Sudoku();
        System.out.println("[Welcome] Welcome to the Sudoku Game.");
        while (true) {
            sudoku.clear();
            String s;
            System.out.println();
            System.out.println("[Select] Please select a mode among:");
            System.out.println("[Select] 1 - Problem Generator");
            System.out.println("[Select] 2 - Problem Solver");
            System.out.println("[Select] 3 - Quit");
            System.out.print("[Select] Your Choice: ");
            s = in.nextLine();
            if (s.equals("1"))
                displayGenerator(sudoku);
            else if (s.equals("2"))
                displaySolver(sudoku);
            else if (s.equals("3"))
                break;
            else
                System.out.println("[Error] Wrong Input.");
        }
        System.out.println("[Quit] Thank you.");
        in.close();
    }

    static void displayGenerator(Sudoku sudoku) {
        while (true) {
            System.out.println("");
            System.out.println("[Note] To efficiently generate problems
with unique solutions, ");
```

```
            System.out.println("[Note] the numbers initially provided must be no less than 22.");
            System.out.println("[Note] Input 0 to quit the generator.");
            System.out.print("[Generator] Input the numbers initially provided: ");
            String s = in.nextLine();
            try {
                int numOfQues = Integer.parseInt(s);
                if (numOfQues == 0)
                    break;
                else if (numOfQues < 22 || numOfQues > 81) {
                    System.out.println("[Error] The input should be between 22 and 81.");
                    continue;
                }
                sudoku.randomInit(numOfQues);
                sudoku.printSudoku();
                System.out.print("[Note] Press Enter again to view the answer.");
                s = in.nextLine();
                sudoku.solve(true, true);
                break;
            } catch (NumberFormatException exception) {
                System.out.println("[Error] Not a number. ");
            }
        }
    }

    static void displaySolver(Sudoku sudoku) {
        while (true) {
            System.out.println("");
            System.out.println("[Note] Replace the blank with one of {'-','_','.','0'}. ");
            System.out.println("[Note] If the solution is not unique,");
            System.out.println("[Note] only one legal solution will be presented.");
            System.out.println("[Solver] Input the problem: ");

            if (sudoku.input(in)) {
                System.out.println("");
                System.out.println("[Solver] Here is the solution: ");
                sudoku.solve(true, true);
                break;
            } else {
                System.out.println("[Error] Wrong Input.");
            }
        }
    }

}

class Sudoku {

    int ans[][];
    int sdk[][];
    int[][] removeSeq = new int[9][];
```

```
    int[] blockRemoved = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };

    Sudoku() {
        ans = new int[9][9];
        sdk = new int[9][9];
        for (int i = 0; i < 9; i++)
            for (int j = 0; j < 9; j++) {
                ans[i][j] = -1;
                sdk[i][j] = -1;
            }
    }

    boolean input(Scanner in) {

        boolean successFlag = true;
        String s;

        for (int i = 0; i < 9; i++) {
            s = in.nextLine();
            if (s == "\n") {
                i--;
            }
            int index = 0;
            int j = 0;
            while (index < s.length()) {
                if (j == 9)
                    break;
                if (s.charAt(index) == '-' || s.charAt(index) == '_' ||
s.charAt(index) == '.') {
                    ans[i][j] = sdk[i][j] = 0;
                    j++;
                } else if (s.charAt(index) >= '0' && s.charAt(index) <=
'9') {
                    ans[i][j] = sdk[i][j] = s.charAt(index) - '0';
                    j++;
                }
                index++;
            }
            if (j != 9)
                break;
        }

        for (int i = 0; i < 9; i++)
            for (int j = 0; j < 9; j++) {
                if (sdk[i][j] == -1)
                    successFlag = false;
            }

        return successFlag;
    }

    boolean check(int pos) {
        int i = pos / 9, j = pos % 9;
        for (int t = 0; t < 9; t++) {
            if (t != i && sdk[t][j] == sdk[i][j])
                return false;
```

```
            if (t != j && sdk[i][t] == sdk[i][j])
                return false;
        }
        int iBlock = i / 3 * 3, jBlock = j / 3 * 3;
        for (int ti = 0; ti < 3; ti++)
            for (int tj = 0; tj < 3; tj++)
                if ((iBlock + ti != i || jBlock + tj != j) &&
                     sdk[iBlock + ti][jBlock + tj] == sdk[i][j])
                    return false;
        return true;
    }

    boolean backtracking(int[][] oper, int pos) {

        boolean found = false;

        if (pos == 81)
            return true;
        int i = pos / 9, j = pos % 9;
        if (ans[i][j] > 0) {
            oper[i][j] = ans[i][j];
            found = backtracking(oper, pos + 1);
        } else
            for (int num = 1; num <= 9; num++) {
                oper[i][j] = num;
                if (check(pos)) {
                    found = backtracking(oper, pos + 1);
                    if (!found)
                        oper[i][j] = 0;
                } else
                    oper[i][j] = 0;
                if (found)
                    break;
            }
        return found;
    }

    boolean solve(boolean toModify, boolean toPrint) {
        int[][] oper;
        if (toModify)
            oper = sdk;
        else
            oper = sdk.clone();
        boolean solved = backtracking(oper, 0);
        if (toPrint) {
            if (solved) {
                System.out.println();
                System.out.println("Solution:");
                System.out.println("\\C 1 2 3 4 5 6 7 8 9");
                System.out.println("R\\ -----------------");

                for (int i = 0; i < 9; i++) {
                    System.out.print(i + 1 + "| ");
                    for (int item : sdk[i]) {
                        if (item > 0)
                            System.out.print(item);
```

```
                        else
                            System.out.print('_');
                        System.out.print(" ");
                    }
                    System.out.println("");
                }
                System.out.println("");
            } else {
                System.out.println("[Failed] No Solution.");
            }
        }
        return solved;
    }

    void printSudoku() {
        System.out.println();
        System.out.println("Sudoku:");
        System.out.println("\\C 1 2 3 4 5 6 7 8 9");
        System.out.println("R\\ -----------------");

        for (int i = 0; i < 9; i++) {
            System.out.print(i + 1 + "| ");
            for (int item : sdk[i]) {
                if (item > 0)
                    System.out.print(item);
                else
                    System.out.print('_');
                System.out.print(" ");
            }
            System.out.println("");
        }
        System.out.println("");
    }

    int[] randomArray(int offset) {
        int[] result = new int[9];
        for (int i = 0; i < 9; i++)
            result[i] = i + offset;
        for (int times = 0; times < 3; times++)
            for (int i = 0; i < 9; i++) {
                int obIndex = (int) (Math.random() * 9);
                int temp = result[obIndex];
                result[obIndex] = result[i];
                result[i] = temp;
            }
        return result.clone();
    }

    void randomInit(int numOfQues) {
        System.out.println("[Init] Initializing the Sudoku with " +
numOfQues + " numbers provided.");
        while (true) {
            this.resetRandomRemoveVar();
            this.randomSet();
            int[] removeBlockSeq = randomArray(0);
            int i = 0;
```

```
            boolean successFlag = true;
            for (i = 0; i < 81 - numOfQues; i++) {
                int tryRemove = 0;
                for (tryRemove = 0; tryRemove < 9; tryRemove++) {
                    if (removeBlockSeq[(i + tryRemove) % 9] == -1) {
                        continue;
                    }
                    if (this.randomRemove(removeBlockSeq[(i + tryRemove) %
9]) == true)
                        break;
                    else
                        removeBlockSeq[(i + tryRemove) % 9] = -1;

                }
                if (tryRemove == 9)
                    successFlag = false;
            }
            if (successFlag) {
                this.reset();
                break;
            }
            this.clear();
        }
    }

    void randomSet() {
        int[] firstCol = randomArray(1);
        for (int i = 0; i < 9; i++)
            ans[i][0] = sdk[i][0] = firstCol[i];
        int[] firstBlock = randomArray(1);
        int current = 0;
        for (int pos : new int[] { 1, 2, 10, 11, 19, 20 }) {
            int i = pos / 9, j = pos % 9;
            while (firstBlock[current] == firstCol[0] ||
firstBlock[current] == firstCol[1]
                    || firstBlock[current] == firstCol[2])
                current++;
            ans[i][j] = sdk[i][j] = firstBlock[current++];
        }
        this.solve(true, false);
        for (int i = 0; i < 9; i++)
            for (int j = 1; j < 9; j++)
                ans[i][j] = sdk[i][j];
        return;
    }

    void resetRandomRemoveVar() {
        for (int i = 0; i < 9; i++) {
            this.removeSeq[i] = randomArray(0);
            this.blockRemoved[i] = 0;
        }
    }

    boolean randomRemove(int block) {
        int origi = block / 3 * 3, origj = block % 3 * 3;
        int deltai, deltaj, i, j;
```

```
312         while (true) {
313             if (blockRemoved[block] == 9)
314                 return false;
315 
316             deltai = removeSeq[block][blockRemoved[block]] / 3;
317             deltaj = removeSeq[block][blockRemoved[block]] % 3;
318             i = origi + deltai;
319             j = origj + deltaj;
320 
321             blockRemoved[block]++;
322             int temp = ans[i][j];
323             for (ans[i][j] = 1; ans[i][j] <= 9; ans[i][j]++) {
324                 if (temp == ans[i][j])
325                     continue;
326                 this.reset();
327                 if (check(i * 9 + j) == false)
328                     continue;
329                 else if (this.solve(false, false) == false)
330                     continue;
331                 else
332                     break;
333 
334             }
335             if (ans[i][j] == 10) {
336                 ans[i][j] = sdk[i][j] = 0;
337                 return true;
338             } else {
339                 ans[i][j] = sdk[i][j] = temp;
340             }
341         }
342     }
343 
344     void clear() {
345         for (int i = 0; i < 9; i++)
346             for (int j = 0; j < 9; j++)
347                 sdk[i][j] = ans[i][j] = -1;
348     }
349 
350     void reset() {
351         for (int i = 0; i < 9; i++)
352             for (int j = 0; j < 9; j++)
353                 sdk[i][j] = ans[i][j];
354     }
355 
356 }
```