面向对象程序设计
综合训练报告

姓名	曹家豪
班级	软件 2204 班
学号	2226114017
电话	13572763245
Email	caojiahao@stu.xjtu.edu.cn
日期	2024-2-21

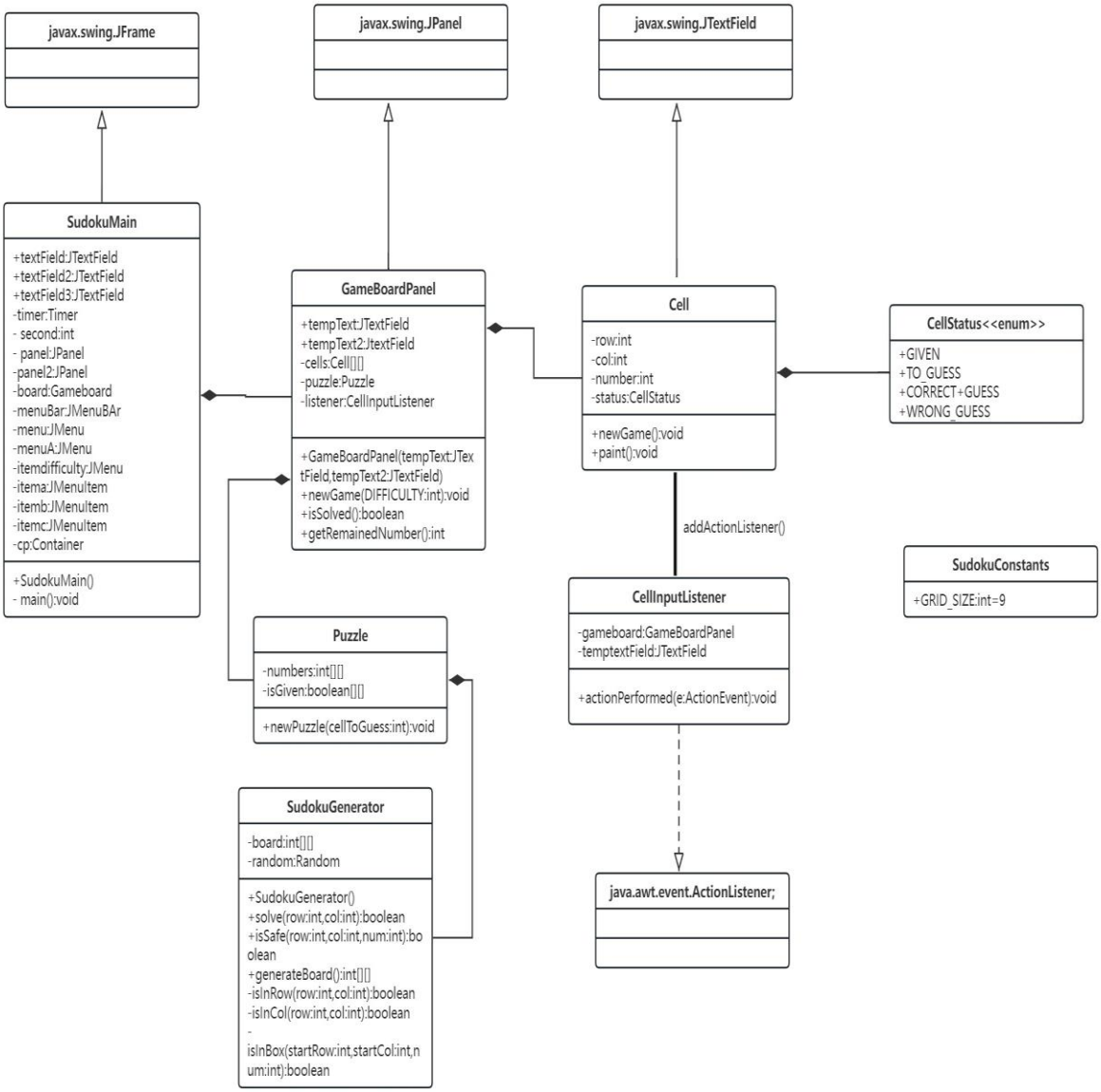
目录

Sudoku

数据设计

以 UML 图来表示该项目的数据设计以及各类之间的关系：

（由于图片压缩较为模糊，上传作业时将 png 文件一并上传）



算法设计

报告中从以下这些角度解释最终数独游戏的实现：

(1) TODO1（为单元格绑定监听器对象）

这个任务主要弄清两点：

①哪些方格需要绑定监听器对象

根据数独规则，只有挖空部分单元格有填充数字的需求，即状态为 *TO_GUESS* 的单元格，那么我们只需要通过**遍历及判断**语句为这些方格绑定能够响应的监听器即可。

②如何为 Swing 组件绑定监听器对象

这是 Swing 框架的基本内容，我们需要创建一个监听器对象（在已有对应类的情况下），然后利用组件对象自带的 *addActionListener* 为其绑定监听器对象。此处我们已有 *CellInputListener* 类（该类的具体定义在后文阐释），则可创建一个响应对象 listener，然后按照①为需要单元格绑定 listener。

(2) TODO2（完善监听器-单元格更新）

这里主要依据回车时填入的数字与默认数字是否一致来更新单元格，当输入数字等于默认数字时，更新单元格状态为 *CORRECT_GUESS*，否则更新*为 WRONG_GUESS*。然后调用 *paint* 方法来重绘单元格（改变单元格颜色）

接着要更新 *textField* 的内容（还剩余多少待完成单元格），调用 *GameBoardPanel* 中定义的 *getRemainedNumber* 方法获取数字，利用 *JTextField* 的 *setText* 方法更新 *textField* 内容。

(3) TODO3（判断对局是否结束）

直接调用 *GameBoardPanel* 中的 *isSolved* 方法判断是否结束，如果游戏结束，调用 *JOptionPane* 的 *showMessageDialog* 方法来显示游戏结束的弹窗。

(4) 自动生成随机数独盘面

定义一个 *SudokuGenerator* 类，其内部的 *GenerateBoard* 方法可以返回一个随机生成的完整数独棋盘，其实现逻辑如下：

定义一个 9x9 的二维数组 board 来表示数独盘面，并初始化一个 Random 随机数生成器。定义 solve 方法，solve 是一个递归函数，用于填充数独格子。从左上角开始，如果当前格子已非空，则尝试填充下一个格子；若为空，则从随机打乱（这是为了能够生成**随机**即每次不同的数独盘面）顺序的 1 到 9 的数字序列中逐一尝试填入。在尝试填入每个数字时，调用 *isSafe* 方法判断这个数字是否安全，即在所在行、列及所在的 3x3 宫格内都不重复出现。如果找到一个安全的数字，将其填入当前格子，并继续递归解决下一个格子。若递归过程中无法找到合法解（即将整个数独填满），则回溯至前一个格子并尝试下一个可能的数字。当所有格子都被合法地填满时，递归结束并返回 true，此时得到的就是一个完整的随机数独盘面。

generateBoard 方法首先调用 *solve* 函数生成数独，如果成功则返回生成的数独盘面，否则抛出异常说明无法生成有效的数独。

(5)游戏前自行选择难度

①难度的选择

在难度选择的实现中，我采用将不同难度设为菜单项加入 *new Game* 的选项中，而游戏的默认难度是 *Easy*。这里需要定义三个 *JMenuItem* 组件分别对应三种难度，将它们添加到 *new Game* 菜单中，为它们绑定事件监听器，能够在点击后开启一轮新游戏，同时根据菜单项不同传入创建新游戏需要的难度参数，从而在初始化棋盘时挖相应数量的空。

②根据不同难度挖不同数目的空

在数组棋盘中，我们采用字节数组来表示是否该位置的单元格需要挖空，那么为了实现不同数目的挖空，我们借用随机数生成 *cellsToGuess*（需要传入 Puzzle 的数据）个不重复的坐标（x，y），再将字节数组中这些坐标下的数据改为 false 即可。

③规定 *Easy* 难度为缺少 5 个空，*Intermediate* 难度为缺少 10 个空，*Difficulty* 难度为缺少 15 个空

(6)菜单栏制作

这里设计 Swing 框架有关菜单的组件知识，*JMenu* 是可以填充子选项二点菜单，*JMenuItem* 是被填充的菜单项，我们依次设置 *File* 菜单对象，*Reset Game* 和 *Exit* 菜单项对象，由于新游戏进行前要选择难度，再设置三个难度的菜单项，添加到 *new Game* 菜单中，将各菜单项用 *add* 方法填充到 *File* 菜单中，定义一个菜单栏对象，利用 *add* 方法将 *File* 填充到菜单栏中，最后，由于数

独主界面 *Container* 采用 *BoardLayout* 布局，我们利用 add 方法将菜单栏填充到主界面北区即可。

(7)未完成单元格数目状态条实现

①获取未完成单元格数目

在 *GameBoardPanel* 类中定义 *getRemainedNumber* 方法，利用 for 循环遍历棋盘，获取状态为 *TO_GUESS* 和 *Wrong_Guess* 的单元格数目。

②创建并初始化状态条

创建一个 *JTextField* 类对象充当状态条，将其放置在主界面南区，由于状态条的初始化应该与新游戏创建同时进行，且内容与游戏难度有关，我们在 *GameBoardPanel* 的 *newGame* 方法中加入状态条的初始化（为了实现该任务，需要 *GameBoardPanel* 增加一个 *JTextField* 数据成员，并在初始化时将主界面的状态条传入）。同时根据 *newGame* 传入的难度参数不同，利用 *setText* 方法使状态条显示不同的内容（*There are xxx puzzles remaining*）。

③状态条的更新

状态条的更新应该在数字填入后进行，那我们可将其实现添加到 *Cell* 的事件监听器中（与之前相同，同样要为 *CellInputListener* 添加一个 *JTextField* 类数据成员并在初始化时传入所需状态条），一旦检测到回车，就调用 *GameBoardPanel* 的 *getRemainedNumber* 方法为状态条更新当前的剩余待完成方格数。

(8)计时器实现

在 *SudokuMain* 类中定义一个整型变量 *seconds* 用于存储经过的秒数。然后创建一个 *Timer* 对象，设置间隔时间为 1000 毫秒（即每秒钟触发一次事件）。

计时器启动后，每当达到设定的间隔时间时，会自动唤醒监听器，调用 *actionPerformed* 方法，该方法内将 *seconds* 加 1，并将实时时间显示在文本框 textField2 上（在主程序中定义，由于主界面空间限制，先将 textField1 与 textField2 添加到以 *FlowLayout* 布局的 *JPanel* 组件中，再将 *JPanel* 组件添加到主界面南侧）。

当用户选择“*New Game*”菜单中的不同难度等级重新开始游戏时，首先会停止计时器，然后重置 *seconds* 为 0，并更新文本框显示新的时间（即 0 秒），最后重新启动计时器。

(9)美化

① 在主界面顶部设置一条文本框，能够显示当前游戏难度，由于顶部已有菜单栏，我们定义一个 *JPanel* 组件，设置其布局管理器为 *BoardLayout*，将菜单栏和该文本框放置在 *JPanel* 组件北区和南区，再将 *JPanel* 组件放置在主界面北区。

在棋盘初始化时依据难度参数设定该文本框内容（类似之前的未完成数状态文本框）

②利用 Swing 框架中组件自带的方法对 *Cell*、*JTextField* 等多个组件设置边框、背景色、字体颜色等。

(10)关于继承、多态等技术

利用 Swing 框架进行 GUI 开发就是对继承、多态等面向对象技术的应用，该实验中的 *Cell* 单元格、棋盘都是对 Swing 原组件的继承，又为了实验内容在定义中增加了新的数据成员。而事件监听器在实现相应接口后，改写其对应方法以完成当前任务需要的相应，很好的体现了多态特性。

此外该项目中将数独棋盘的大小等参数放在一个单独的类中，在 *newGame* 方法中定义大多行为，在主程序中只需要在调用时传入不同参数，这些行为也提高了代码的复用率与可扩展性。

主干代码说明

主要的实现逻辑已在上文说明，这里选取部分程序解释具体的代码实现：

①棋盘生成

```
1. // 定义一个名为SudokuGenerator 的类，用于生成数独盘面
2. public class SudokuGenerator {
3.     // 声明常量SIZE 表示数独盘面的大小（9x9）
4.     private static final int SIZE = 9;
5.
6.     // 定义一个二维数组board 来存储数独盘面数据
7.     private int[][] board;
8.
9.     // 创建一个Random 对象，用于生成随机数
10.    private Random random = new Random();
11.
12.    // 构造函数，初始化 9x9 的空白数独盘面
13.    public SudokuGenerator() {
14.        board = new int[SIZE][SIZE];
15.    }
16.
```

```
17.     // solve 方法：递归填充数独格子
18.     // 参数 row 和 col 分别表示当前处理的行和列索引
19.     public boolean solve(int row, int col) {
20.         // 当一行填完时，换到下一行的首列；当最后一列填完时，说明数独已解决
21.         if (row == SIZE) {
22.             col++;
23.             if (col == SIZE) {
24.                 return true; // 解决完成
25.             }
26.             row = 0;
27.         }
28.
29.         // 如果当前位置已有数字，则递归处理下一个位置
30.         if (board[row][col] != 0) {
31.             return solve(row + 1, col);
32.         }
33.
34.         // 创建一个包含 1 至 9 的整数列表，并对其进行随机打乱顺序
35.         List<Integer> numbers = new ArrayList<>();
36.         for (int i = 1; i <= SIZE; i++) {
37.             numbers.add(i);
38.         }
39.         Collections.shuffle(numbers, random); // 随机化数字顺序
40.
41.         // 尝试用随机顺序中的每个数字填入当前空格
42.         for (int num : numbers) {
43.             // 检查填入当前格子是否安全（不在同一行、同一列及同一宫格内出现）
44.             if (isSafe(row, col, num)) {
45.                 // 若安全则填入该数字
46.                 board[row][col] = num;
47.
48.                 // 递归解决下一个格子，若能成功解决整个数独则返回 true
49.                 if (solve(row + 1, col)) {
50.                     return true;
51.                 }
52.
53.                 // 若无法通过当前数字解出完整数独，则回溯，将当前格子置零重新尝试
54.                 board[row][col] = 0; // 回溯操作
55.             }
56.         }
```

```
57.
58.         // 当所有可能数字都无法安全填入时，返回false 表示当前路径失败
59.         return false;
60.     }
61.
62.     // isSafe 方法：判断在给定的位置(row, col)填入 num 是否合法
63.     public boolean isSafe(int row, int col, int num) {
64.         // 不在同一行、同一列及同一 3x3 宫格内的条件下才为安全
65.         return !isInRow(row, num) && !isInCol(col, num) && !isInBox(row - row % 3, col - col % 3, num);
66.     }
67.
68.     // isInRow 方法：检查给定的数字 num 是否出现在指定行(row)
69.     private boolean isInRow(int row, int num) {
70.         for (int d = 0; d < SIZE; d++) {
71.             if (board[row][d] == num) {
72.                 return true;
73.             }
74.         }
75.         return false;
76.     }
77.
78.     // isInCol 方法：检查给定的数字 num 是否出现在指定列(col)
79.     private boolean isInCol(int col, int num) {
80.         for (int r = 0; r < SIZE; r++) {
81.             if (board[r][col] == num) {
82.                 return true;
83.             }
84.         }
85.         return false;
86.     }
87.
88.     // isInBox 方法：检查给定的数字 num 是否出现在指定 3x3 宫格区域(startRow, startCol)
89.     private boolean isInBox(int startRow, int startCol, int num) {
90.         for (int row = 0; row < 3; row++) {
91.             for (int col = 0; col < 3; col++) {
92.                 if (board[row + startRow][col + startCol] == num) {
93.                     return true;
94.                 }
95.             }
96.         }
```

```
97.         return false;
98.     }
99.
100.        // generateBoard 方法：生成完整的数独盘面并返回
101.        public int[][] generateBoard() {
102.            // 调用 solve 方法从(0, 0)开始填充数独
103.            if (solve(0, 0)) {
104.                // 解决成功，返回生成的数独盘面
105.                return board;
106.            } else {
107.                // 解决失败，抛出异常表明未能生成有效的数独盘面
108.                throw new IllegalStateException("无法生成有效
    的数独盘面。");
109.            }
110.        }
111.    }
112.
```

Ctrl+M

②棋盘挖空

```
1. // 创建一个 ArrayList 来存储随机生成的位置数组，每个位置数组包含
   一个数独单元格的行和列坐标
2. List<int[]> randomPositions = new ArrayList<>();
3.
4. // 创建一个 Random 对象用于生成随机数
5. Random rand = new Random();
6.
7. // 创建一个布尔型数组 usedPositions，大小为数独总单元格数量
   （9x9），初始化所有元素为 false，表示这些位置尚未使用过
8. boolean[] usedPositions = new boolean[SudokuConstants.GRID_
   SIZE * SudokuConstants.GRID_SIZE];
9.
10.// 当已生成的随机位置数量小于需要猜测的单元格数量时，持续循环
11.while (randomPositions.size() < cellsToGuess) {
12.    // 随机生成一个数独行坐标（范围在 0 到 8 之间）
13.    int row = rand.nextInt(9);
14.
15.    // 随机生成一个数独列坐标（范围在 0 到 8 之间）
16.    int col = rand.nextInt(9);
17.
18.    // 计算当前选定单元格对应的数组索引（行列坐标转换为一维索引）
19.    int index = row * 9 + col;
20.
21.    // 检查该位置是否已被使用过，如果未被使用，则进行以下操作
```

```
22.     if (!usedPositions[index]) {
23.         // 将该位置标记为已使用
24.         usedPositions[index] = true;
25.
26.         // 将当前生成的随机行、列坐标封装成一个数组，并添加至随机位置列表中
27.         randomPositions.add(new int[]{row, col});
28.     }
29.}
30.
```

Ctrl+M

③计时器实现

```
1. Timer timer = new Timer(1000, new ActionListener() {
2.
3.             @Override
4.             public void actionPerformed(ActionEvent e) {
5.                 seconds++; // 增加秒数
6.                 textField2.setText("Time:"+ String.valueOf(seconds)); // 更新文本字段
7.             }
8.         });
9.         // 启动计时器
10.        timer.start();
11.JMenuItem itema=new JMenuItem("Easy");
12.        itema.addActionListener(new ActionListener(){
13.            @Override
14.            public void actionPerformed(ActionEvent e){
15.                board.newGame(1);
16.                timer.stop();
17.                // 重置秒数
18.                seconds = 0;
19.                // 更新文本字段显示的时间
20.                textField2.setText("Time:"+String.valueOf(seconds));
21.                // 重新启动计时器
22.                timer.start();
23.            }
24.        });
```

Ctrl+M

运行结果展示

（运行结果的录屏另外上传）

总结和收获

①学习了 java 语言利用 Swing 框架进行 GUI 开发的基础知识

②通过数独项目的完成，更加体会到继承、多态等面向对象特性以及提高代码复用率的重要性，也是对学习 GUI 开发知识的一次应用。

分值表

功能列表	打分说明	分值
扩展 1	能够实时显示游戏的状态信息为 7 分	7
扩展 2	可以正确将时间显示在扩展 1 的状态条中显示为 7 分	6
扩展 3	和运行原型的界面效果有变化即可加 2 分，在此基础上视觉效果更 赏心悦目加 1-4 分。	4
填表人：曹家豪		

附录

```
// 创建一个ArrayList来存储随机生成的位置数组，每个位置数组包含一个数独单元格的行和列坐标
List<int[]> randomPositions = new ArrayList<>();
// 创建一个Random对象用于生成随机数
Random rand = new Random();
// 创建一个布尔型数组usedPositions，大小为数独总单元格数量（9x9），初始化所有元素为false，表示这些位置尚未使用过
boolean[] usedPositions = new boolean[SudokuConstants.GRID_SIZE * SudokuConstants.GRID_SIZE];
// 当已生成的随机位置数量小于需要猜测的单元格数量时，持续循环
while (randomPositions.size() < cellsToGuess) {
    // 随机生成一个数独行坐标（范围在0到8之间）
    int row = rand.nextInt(9);
    // 随机生成一个数独列坐标（范围在0到8之间）
    int col = rand.nextInt(9);
    // 计算当前选定单元格对应的数组索引（行列坐标转换为一维索引）
    int index = row * 9 + col;
    // 检查该位置是否已被使用过，如果未被使用，则进行以下操作
    if (!usedPositions[index]) {
        // 将该位置标记为已使用
        usedPositions[index] = true;
        // 将当前生成的随机行、列坐标封装成一个数组，并添加至随机位置列表中
        randomPositions.add(new int[]{row, col});
    }
}
package sudoku;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
/*
 * 创建所有单元格都可以使用的一个监听器类型
 */
public class CellInputListener implements ActionListener {
    private GameBoardPanel gameboard;
    JTextField temptextField;
    public CellInputListener(GameBoardPanel gameboard, JTextField textField){
        if (gameboard == null)
            throw new IllegalArgumentException("Null pointer reference.");
```

```
37.         this.gameboard = gameboard;
38.         temptextField = textField;
39.     }
40.     @Override
41.     public void actionPerformed(ActionEvent e) {
42.         // 获得是哪个单元格出发了回车事件（获得事件源）
43.         Cell sourceCell = (Cell)e.getSource();
44.
45.         // 获得输入的数字
46.         int numberIn = Integer.parseInt(sourceCell.getText(
   ));
47.
48.         /*
49.          * [TODO 2]
50.          * 检查发生回车敲击事件的单元格中存储的数字和用户输入的
   数字是否相等
51.          * 根据上面的判断结论更新单元格的状态为
   CellStatus.CORRECT_GUESS 或者 CellStatus.WRONG_GUESS
52.          * 一旦单元格的属性值（状态就是它的属性值之一）被更新，
   那么就应该调用 sourceCell.paint()，触发重新绘制单元格的外观
53.          */
54.         boolean isright = (numberIn == sourceCell.number);
55.         if(isright){
56.             sourceCell.status = CellStatus.CORRECT_GUESS;
57.             sourceCell.paint();
58.
59.         }else{
60.             sourceCell.status = CellStatus.WRONG_GUESS;
61.             sourceCell.paint();
62.         }
63.         temptextField.setText("There are "+gameboard.getRem
   ainedNumber()+" puzzles remaining.");
64.         /*
65.          * [TODO 3]
66.          * 一个单元格的状态变化了，那么就应该调用
   GameBoardPanel 中的 isSolved 方法，用来判断游戏是否结束
67.          * 如果游戏成功解决，那么就可以弹出对话框显示结果。
68.          */
69.         if(gameboard.isSolved()){
70.             JPanel panel = new JPanel();
71.             JLabel label = new JLabel("Congratulations! You
    have solved the puzzle!");
72.             panel.add(label);
73.
```

```
74.             JOptionPane.showMessageDialog(null, panel, "Res
    ult", JOptionPane.INFORMATION_MESSAGE);
75.         }
76.     }
77.}
78.package sudoku;
79./**
80. * 单元格在游戏中会有不同的状态。这些状态可以使用：
81. * 1.整型变量表示（比如 1、2），但是这样表示状态可读性不高；
82. * 2.字符串值表示（比如“correct”等）,这样表示状态则在比较时需要
    考虑字符串的大小写等问题。
83. * 在类似的场景中，建议定义枚举类型完成此类工作，枚举类型的可读性
    高，且是类型安全。
84. */
85.public enum CellStatus {
86.     GIVEN,          // 具有此状态的单元格表示其包含的数字是预先
    给定的；
87.     TO_GUESS,       // 具有此状态的单元格是不包含任何数字，需要
    游戏玩家进行填充的；
88.     CORRECT_GUESS, // 具有此状态的单元格表明游戏玩家正确填充了
    其应该包含的数字；
89.     WRONG_GUESS     // 具有此状态的单元格表明游戏玩家错误填充了
    其应该包含的数字。
90.     // 一场游戏是否成功结束的判断条件就是没有任何一个单元格的状态
    是 TO_GUESS 或者 WRONG_GUESS
91.}
```

Ctrl+M

```
1. package sudoku;
2. import java.awt.*;
3. import javax.swing.*;
4.
5. /*
6.  * 创建一个 JPanel 的子类类型，该面板中包含了 9*9 个 Cell 对象，采
    用了 GridLayout 布局管理器
7.  */
8. public class GameBoardPanel extends JPanel {
9.     private static final long serialVersionUID = 1L;
10.
11.     // 以像素单位为每一个 Cell 单元格设置外观大小
12.     public static final int CELL_SIZE = 60;
13.     public static final int BOARD_WIDTH  = CELL_SIZE * Sudo
    kuConstants.GRID_SIZE;
14.     public static final int BOARD_HEIGHT = CELL_SIZE * Sudo
    kuConstants.GRID_SIZE;
```

```
15.
16.
17.     /** 该游戏面板是由9x9个Cell对象 (Cell是JTextFields的子类)构成 */
18.     private Cell[][] cells = new Cell[SudokuConstants.GRID_SIZE][SudokuConstants.GRID_SIZE];
19.     private Puzzle puzzle = new Puzzle();
20.     private CellInputListener listener;
21.     JTextField tempText;
22.     JTextField tempText2;
23.
24.     public GameBoardPanel(JTextField tempText, JTextField tempText2) {
25.         super.setLayout(new GridLayout(SudokuConstants.GRID_SIZE, SudokuConstants.GRID_SIZE));
26.         this.tempText = tempText;
27.         this.tempText2 = tempText2;
28.         // 将Cell对象组件加入到Panle对象中
29.         for (int row = 0; row < SudokuConstants.GRID_SIZE; ++row) {
30.             for (int col = 0; col < SudokuConstants.GRID_SIZE; ++col) {
31.                 cells[row][col] = new Cell(row, col);
32.                 cells[row][col].setBorder(BorderFactory.createLineBorder(Color.GRAY,5));
33.                 super.add(cells[row][col]);
34.             }
35.         }
36.         listener = new CellInputListener(this,tempText2);
37.
38.
39.         // 设置面板的大小
40.         super.setPreferredSize(new Dimension(BOARD_WIDTH, BOARD_HEIGHT));
41.     }
42.
43.     /**
44.      * 生成一局新游戏，基于生成的新游戏中的数据重新初始化 gameboard 中包含的每一个 cell 对象
45.      */
46.     public void newGame(int DIFFICUTY) {
47.         if(DIFFICUTY == 1){
48.             puzzle.newPuzzle(5);
49.             tempText.setText("Degree of difficulty: Easy");
```

```
50.             tempText2.setText("There are 5 puzzles remainin
    g.");
51.         }
52.         else if(DIFFICUTY == 2){
53.             puzzle.newPuzzle(10);
54.             tempText.setText("Degree of difficulty: Interme
    diate");
55.             tempText2.setText("There are 10 puzzles remaini
    ng.");
56.         }
57.         else if(DIFFICUTY == 3){
58.             puzzle.newPuzzle(15);
59.             tempText.setText("Degree of difficulty: Difficu
    lty");
60.             tempText2.setText("There are 15 puzzles remaini
    ng.");
61.         }else{
62.             puzzle.newPuzzle(5);
63.             tempText.setText("Degree of difficulty: Easy");
64.             tempText2.setText("There are 5 puzzles remainin
    g.");
65.         }
66.         for (int row = 0; row < SudokuConstants.GRID_SIZE;
    ++row) {
67.             for (int col = 0; col < SudokuConstants.GRID_SI
    ZE; ++col) {
68.                 cells[row][col].newGame(puzzle.numbers[row]
    [col], puzzle.isGiven[row][col]);
69.             }
70.         }
71.
72.         // [TODO 1]
73.         // 为所有可编辑的单元格（即需要输入数字）绑定监听器对象
74.         for (int row = 0; row < SudokuConstants.GRID_SIZE;
    ++row) {
75.             for (int col = 0; col < SudokuConstants.GRID_SI
    ZE; ++col) {
76.                 if (cells[row][col].status == CellStatus.TO
    _GUESS) {
77.                     cells[row][col].addActionListener(liste
    ner);
78.                 }
79.             }
80.         }
81.     }
```

```
82.
83.     /**
84.      * 如果当局游戏成功解决，那么返回 true
85.      * 判断依据：只要有一个单元格的状态是 CellStatus.TO_GUESS 或者 CellStatus.WRONG_GUESS，那么游戏就没有结束。
86.      */
87.     public boolean isSolved() {
88.         for (int row = 0; row < SudokuConstants.GRID_SIZE; ++row) {
89.             for (int col = 0; col < SudokuConstants.GRID_SIZE; ++col) {
90.                 if (cells[row][col].status == CellStatus.TO_GUESS || cells[row][col].status == CellStatus.WRONG_GUESS) {
91.                     return false;
92.                 }
93.             }
94.         }
95.         return true;
96.     }
97.     public int getRemainedNumber(){
98.         int count = 0;
99.         for (int row = 0; row < SudokuConstants.GRID_SIZE; ++row) {
100.             for (int col = 0; col < SudokuConstants.GRID_SIZE; ++col) {
101.                 if (cells[row][col].status == CellStatus.TO_GUESS|| cells[row][col].status == CellStatus.WRONG_GUESS) {
102.                     count++;
103.                 }
104.             }
105.         }
106.         return count;
107.     }
108. }
109. package sudoku;
110.
111. import java.util.ArrayList;
112. import java.util.List;
113. import java.util.Random;
114.
115. /**
116.  * 定义 Puzzle 类型，用来表达和一局游戏相关的信息。
117.  */
```

```
118.   public class Puzzle {
119.       //numbers 数组存储一局游戏的数字分布
120.       int[][] numbers = new int[SudokuConstants.GRID_SIZE]
   [SudokuConstants.GRID_SIZE];
121.       // isGiven 数组用来存储一局游戏单元格的数字是否暴露的状
   态
122.       boolean[][] isGiven = new boolean[SudokuConstants.GR
   ID_SIZE][SudokuConstants.GRID_SIZE];
123.
124.       // 根据设定需要猜测的单元个数新生成一局独数
125.       // 可以利用猜测的单元个数的多少做为游戏难度级别的设定依
   据
126.       // 这个方法需要对 numbers 数组和 isGiven 数组进行更新
127.       public void newPuzzle(int cellsToGuess) {
128.           // hardcodedNumbers 是预先设定的一局游戏的数字分布
129.           SudokuGenerator sudokuGenerator = new SudokuGene
   rator();
130.           int[][] hardcodedNumbers = sudokuGenerator.gener
   ateBoard();
131.           for (int row = 0; row < SudokuConstants.GRID_SIZ
   E; ++row) {
132.               for (int col = 0; col < SudokuConstants.GRID
   _SIZE; ++col) {
133.                   numbers[row][col] = hardcodedNumbers[row
   ][col];
134.               }
135.           }
136.
137.           // 可以使用 cellsToGuess 的值初始化 isGiven 数组中
   false 的数量，即需要猜测的单元格数量
138.           // hardcodedIsGiven 是预先设定的有哪些位置的单元格
   被暴露（在下面的数据中只指定暴露 2 个单元格）
139.           boolean[][] hardcodedIsGiven = {
140.                   {true, true, true, true, true, t
   rue, true, true, true},
141.                   {true, true, true, true, true, t
   rue, true, true, true},
142.                   {true, true, true, true, true, t
   rue, true, true, true},
143.                   {true, true, true, true, true, t
   rue, true, true, true},
144.                   {true, true, true, true, true, t
   rue, true, true, true},
145.                   {true, true, true, true, true, t
   rue, true, true, true},
```

```
                                {true, true, true, true, true, true, true, true, true},
                                {true, true, true, true, true, true, true, true, true},
                                {true, true, true, true, true, true, true, true, true}};
            List<int[]> randomPositions = new ArrayList<>();
            Random rand = new Random();
            boolean[] usedPositions = new boolean[SudokuConstants.GRID_SIZE*SudokuConstants.GRID_SIZE];
            while (randomPositions.size() < cellsToGuess) {
                int row = rand.nextInt(9);
                int col = rand.nextInt(9);
                int index = row * 9 + col;
                if (!usedPositions[index]) {
                    usedPositions[index] = true;
                    randomPositions.add(new int[]{row, col});
                }
            }
            for (int[] pos : randomPositions) {
                hardcodedIsGiven[pos[0]][pos[1]] =false;}

            for (int row = 0; row < SudokuConstants.GRID_SIZE; ++row) {
                for (int col = 0; col < SudokuConstants.GRID_SIZE; ++col) {
                    isGiven[row][col] = hardcodedIsGiven[row][col];
                }
            }
        }
    }
```

Ctrl+M

```
package sudoku;
/**
 * 为当前应用中很多类需要使用的常量完成定义
 * 将逻辑上相关联的一组类所需要的常量定义在一个类中是可以借鉴的，这样易于维护。
 */
public class SudokuConstants {
    /** 定义游戏面板的大小，针对当前游戏是指每一行的单元格数 */
    public static final int GRID_SIZE = 9;
}
```

```java
package sudoku;
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;
import java.util.Random;

public class SudokuGenerator {
    private static final int SIZE = 9;
    private int[][] board;
    private Random random = new Random();

    public SudokuGenerator() {
        board = new int[SIZE][SIZE];
    }

    public boolean solve(int row, int col) {
        if (row == SIZE) {
            col++;
            if (col == SIZE) {
                return true; // Puzzle solved
            }
            row = 0;
        }

        if (board[row][col] != 0) {
            return solve(row + 1, col);
        }

        List<Integer> numbers = new ArrayList<>();
        for (int i = 1; i <= SIZE; i++) {
            numbers.add(i);
        }
        Collections.shuffle(numbers, random); // Randomize numbers

        for (int num : numbers) {
            if (isSafe(row, col, num)) {
                board[row][col] = num;
                if (solve(row + 1, col)) {
                    return true;
                }
                board[row][col] = 0; // Backtrack
            }
        }

```

```
54.         return false;
55.     }
56.
57.     public boolean isSafe(int row, int col, int num) {
58.         return !isInRow(row, num) && !isInCol(col, num) &&
   !isInBox(row - row % 3, col - col % 3, num);
59.     }
60.
61.     private boolean isInRow(int row, int num) {
62.         for (int d = 0; d < SIZE; d++) {
63.             if (board[row][d] == num) {
64.                 return true;
65.             }
66.         }
67.         return false;
68.     }
69.
70.     private boolean isInCol(int col, int num) {
71.         for (int r = 0; r < SIZE; r++) {
72.             if (board[r][col] == num) {
73.                 return true;
74.             }
75.         }
76.         return false;
77.     }
78.
79.     private boolean isInBox(int startRow, int startCol, int
    num) {
80.         for (int row = 0; row < 3; row++) {
81.             for (int col = 0; col < 3; col++) {
82.                 if (board[row + startRow][col + startCol] =
   = num) {
83.                     return true;
84.                 }
85.             }
86.         }
87.         return false;
88.     }
89.     public int[][] generateBoard() {
90.         if (solve(0, 0)) {
91.             return board;
92.         } else {
93.             throw new IllegalStateException("Unable to gene
   rate a valid Sudoku board.");
94.         }
```

```
95.     }
96.}
97.package sudoku;
98.
99.import java.awt.*;
100.    import java.awt.event.ActionEvent;
101.    import java.awt.event.ActionListener;
102.    import javax.swing.*;
103.    /**
104.     * 运行 Sudoku 游戏的程序
105.     */
106.    public class SudokuMain extends JFrame {
107.        static final int DefaultDifficuty=1;
108.        private static final long serialVersionUID = 1L;
109.        private int seconds = 0;
110.        public SudokuMain() {
111.            JPanel panel = new JPanel();
112.            JPanel panel2 = new JPanel();
113.            panel2.setLayout(new BorderLayout());
114.            panel.setLayout(new FlowLayout(FlowLayout.CENTER
    , 10, 2));
115.            JTextField textField = new JTextField(40);
116.            textField.setBackground(Color.CYAN);
117.            textField.setForeground(Color.blue);
118.            JTextField textField2 = new JTextField(40);
119.            textField2.setBackground(Color.CYAN);
120.            Font timeFont = new Font("Arial", Font.BOLD, 20)
    ;
121.            textField.setFont(timeFont);
122.            textField2.setFont(timeFont);
123.            JTextField textField3 = new JTextField();
124.            textField3.setHorizontalAlignment(JTextField.CEN
    TER);
125.            textField3.setBackground(Color.gray);
126.            Font dffont = new Font("Arial", Font.BOLD, 30);
127.            textField3.setFont(dffont);
128.            textField3.setForeground(Color.pink);
129.            Timer timer = new Timer(1000, new ActionListener
    () {
130.
131.                @Override
132.                public void actionPerformed(ActionEvent e) {
133.                    seconds++; // 增加秒数
134.                    textField2.setText("Time:"+ String.value
    Of(seconds)); // 更新文本字段
```

```
135.                 }
136.             });
137.             // 启动计时器
138.             timer.start();
139.             panel.add(textField);
140.             panel.add(textField2);
141.             textField.setHorizontalAlignment(JTextField.CENT
    ER);
142.             textField2.setHorizontalAlignment(JTextField.CEN
    TER);
143.             GameBoardPanel board = new GameBoardPanel(textFi
    eld3,textField);
144.
145.             board.setBorder(BorderFactory.createEmptyBorder(
    10, 10, 10, 10));
146.
147.             JMenuBar menuBar = new JMenuBar();
148.             JMenu menu = new JMenu("File");
149.             JMenu itemA = new JMenu("New Game");
150.             JMenu itemdifficuty=new JMenu("Difficulty");
151.             JMenuItem itema=new JMenuItem("Easy");
152.             itema.addActionListener(new ActionListener(){
153.                 @Override
154.                 public void actionPerformed(ActionEvent e){
155.                     board.newGame(1);
156.                     timer.stop();
157.                     // 重置秒数
158.                     seconds = 0;
159.                     // 更新文本字段显示的时间
160.                     textField2.setText("Time:"+String.valueO
    f(seconds));
161.                     // 重新启动计时器
162.                     timer.start();
163.                 }
164.             });
165.             JMenuItem itemb = new JMenuItem("Intermediate");
166.             itemb.addActionListener(new ActionListener(){
167.                 @Override
168.                 public void actionPerformed(ActionEvent e){
169.                     board.newGame(2);
170.                     timer.stop();
171.                     // 重置秒数
172.                     seconds = 0;
173.                     // 更新文本字段显示的时间
```

```
174.                     textField2.setText("Time:"+String.valueO
    f(seconds));
175.                     // 重新启动计时器
176.                     timer.start();
177.                 }
178.             });
179.             JMenuItem itemc = new JMenuItem("Difficult");
180.             itemc.addActionListener(new ActionListener(){
181.                 @Override
182.                 public void actionPerformed(ActionEvent e){
183.                     board.newGame(3);
184.                     timer.stop();
185.                     // 重置秒数
186.                     seconds = 0;
187.                     // 更新文本字段显示的时间
188.                     textField2.setText("Time:"+String.valueO
    f(seconds));
189.                     // 重新启动计时器
190.                     timer.start();
191.                 }
192.             });
193.             itemdifficuty.add(itema);
194.             itemdifficuty.add(itemb);
195.             itemdifficuty.add(itemc);
196.             itemA.add(itemdifficuty);
197.             JMenuItem itemB = new JMenuItem("Reset Game");
198.
199.             JMenuItem itemC = new JMenuItem("Exit");
200.             itemC.addActionListener(new ActionListener(){
201.                 @Override
202.                 public void actionPerformed(ActionEvent e){
203.                     System.exit(0);
204.                 }
205.             });
206.             menu.add(itemA);
207.             menu.add(itemB);
208.             menu.add(itemC);
209.             menuBar.add(menu);
210.
211.             panel2.add(menuBar, BorderLayout.NORTH);
212.             panel2.add(textField3, BorderLayout.SOUTH);
213.             Container cp = getContentPane();
214.
215.             cp.setLayout(new BorderLayout());
216.             cp.add(panel2, BorderLayout.NORTH);
```

```
217.            cp.add(board, BorderLayout.CENTER);
218.            cp.add(panel, BorderLayout.SOUTH);
219.
220.            // 初始化一局游戏面板
221.            board.newGame(DefaultDifficuty);
222.
223.            pack();
224.            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
225.            setTitle("Sudoku");
226.            setVisible(true);
227.        }
228.
229.        public static void main(String[] args) {
230.            SwingUtilities.invokeLater(new Runnable(){
231.                @Override
232.                public void run(){
233.                    new SudokuMain();
234.                }
235.            });
236.        }
237.    }
```

Ctrl+M