面向对象程序设计
综合训练实验报告

第一次

姓名

班级

学号

电话

Email

日期

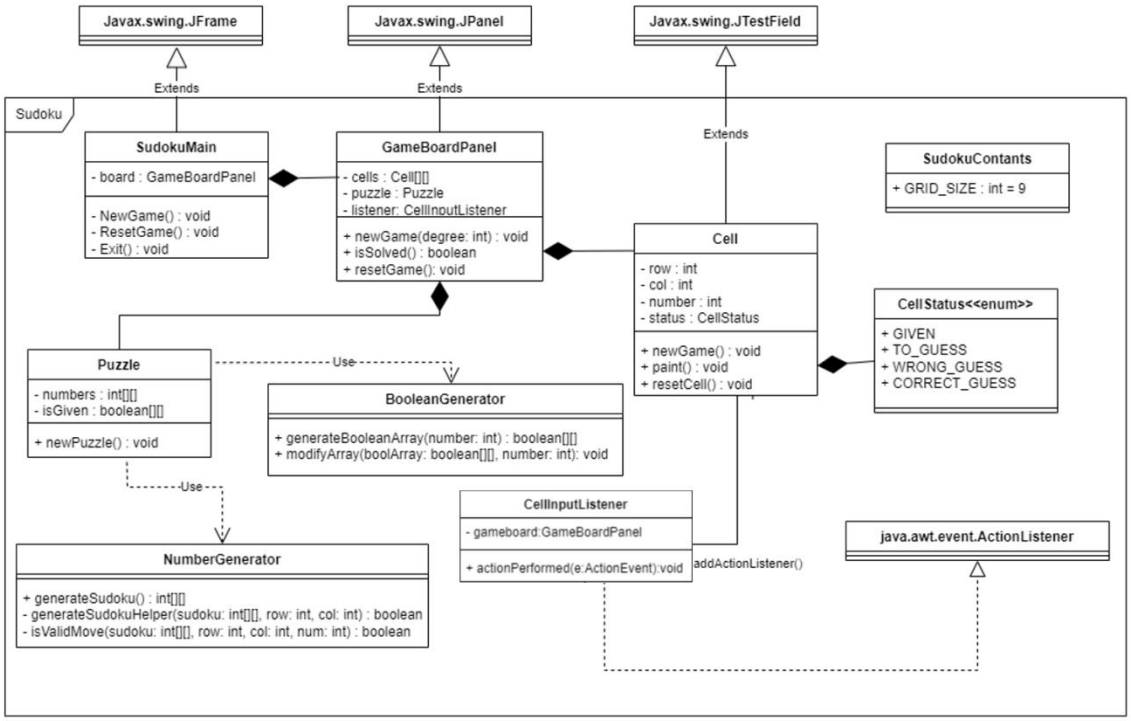
目录

基础部分

扩展部分

基础部分

基础部分 UMI 图:



改动部分

一、Cell 类:

程序设计及代码说明:

1.新增 resetCell（）方法，用于初始化单元格:

（1）将单元格的状态 status 改为 CellStatus.To_GUESS;

（2）调用 setTest（）方法将该单元格中的文本设置为空;

（3）调用 setEditable（true）方法将其设为可见;

（4）调用 setBackground（BG_TO_GUESS）方法将单元格的背景颜色改为带猜测的单元格应有的背景颜色;

（5）调用 setForeground（FG_NOT_GIVEN）方法修改编辑该单元格时的字体颜色。

新增部分代码展示：

```
1 usage
public void resetCell() {
    //将单元格初始化
    status = CellStatus.TO_GUESS;
    setText("");
    setEditable(true);
    setBackground(BG_TO_GUESS);
    setForeground(FG_NOT_GIVEN);
}
```

二、Puzzle 类：

程序设计及代码说明：

1.改动 newPuzzle（int cellsToGuess）方法，将其改为有参函数，接收整数类型参数 cellsToGuess：

（1）调用 NumberGenerator 中的 getNumbers（）方法给成员变量 numbers 赋值，即为随机生成的 9*9 大小的二维数组（NumberGenerator 类是新添的辅助类，仅用于随机生成 9*9 大小的二维数组，其设计思想和代码均在后续“新添部分”展示）；

（2）同理调用 BooleanGenerator 中的 getBoolean（int cellsToGuess）方法，将 newPuzzle 方法接收的参数 cellsToGuess 传入 getBoolean 方法中，根据参数的值来随机生成对应的二维布尔类型数组。

修改部分代码展示：

```
1 usage
public void newPuzzle(int cellsToGuess) {
    //随机生成一个符合数独要求的数组
    numbers = NumberGenerator.getNumbers();

    //根据难度随机生成待猜测的单元格数量
    isGiven = BooleanGenerator.getBoolean(cellsToGuess);
}
```

三、GameBoardPanel 类：

程序设计及代码说明：

1. 完成 TODO1：

调用两层 for 循环，给 9*9 大小的二维数组 cells 中的每一个单元格做调整：先用 if 语句判断该单元格的状态 status 是否为待猜测单元格的状态 TO_GUESS，若是则调用 addActionListener（listener）方法为该单元格添加监听器对象。

2. 新增 resetGame（）方法：

调用两层 for 循环，对二维数组 cells 中的每一个单元格的状态 status 进行检索，调用 if 语句判断其状态不为 GIVEN 时，对该单元格调用 resetCell（）方法将其初始化。

新增代码展示：

（1）TODO1 添加的代码：

```
/**
 * 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
 */
2 usages
public void newGame(int degree) {
    puzzle.newPuzzle(degree);
    for (int row = 0; row < SudokuConstants.GRID_SIZE; row++) {
        for (int col = 0; col < SudokuConstants.GRID_SIZE; col++) {
            cells[row][col].newGame(puzzle.numbers[row][col], puzzle.isGiven[row][col]);
        }
    }
    //[TODO 1]
    // 为所有可编辑的单元格（即需要输入数字）绑定监听器对象
    for (int row = 0; row < SudokuConstants.GRID_SIZE; row++) {
        for (int col = 0; col < SudokuConstants.GRID_SIZE; col++) {
            if (cells[row][col].status.equals(CellStatus.TO_GUESS)) {
                cells[row][col].addActionListener(listener);
            }
        }
    }
}
```

（2）新添 resetGame（）方法的代码:

```
1 usage
public void resetGame() {
    // 重置所有可编辑的单元格的数字和状态
    for (int row = 0; row < SudokuConstants.GRID_SIZE; row++) {
        for (int col = 0; col < SudokuConstants.GRID_SIZE; col++) {
            if (cells[row][col].status != CellStatus.GIVEN) {
                cells[row][col].resetCell();
            }
        }
    }
}
```

四、CellInputListener 类:

程序设计及代码说明:

1. 完成 TODO2:

（1）用 if 语句判断用户输入的数字 numberIn 是否等于该数独盘面在该位置的数字大小，若相等则修改当前单元格的状态 status 为 CORRECT_GUESS，若用户输入错误的数字则将单元格状态改为 WRONG_GUESS;

（2）调用 paint（）方法进行刷新。

2. 完成 TODO3：

（1）调用 if 语句调用 isSolved（）方法判断 gameboard 棋盘上的所有待填入的单元格是否已经全部被正确填充，若是，则判定游戏结束：

调用 JOptionPane 类中的 showOptionDialog 方法，并用整数类型变量 option 接收；

调用 if 语句判断 option 是否等于 JOptionPane.OK_OPTION，若是则调用 System.exit（0）方法退出数独游戏。

新增代码展示：

（1）TODO2 添加的代码：

```
/*
 * [TODO 2]
 * 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
 * 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
 * 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
 */
if (numberIn == sourceCell.number) {
    sourceCell.status = CellStatus.CORRECT_GUESS;
} else {
    sourceCell.status = CellStatus.WRONG_GUESS;
}
sourceCell.paint();
```

（2）TODO3 添加的代码：

```
/*
 * [TODO 3]
 * 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
 * 如果游戏成功解决，那么就可以弹出对话框显示结果。
 */
if (gameboard.isSolved()) {
    // 游戏解决，显示成功对话框
    int option = JOptionPane.showOptionDialog( parentComponent: null,  message: "恭喜您挑战成功！\n点击确定退出游戏",
            title: "游戏结束", JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE,
            icon: null, new Object[]{"确定"},  initialValue: "确定");

    // 当用户点击确定按钮时，退出程序
    if (option == JOptionPane.OK_OPTION) {
        System.exit( status: 0);
    }
}
```

五、SudokuMain 类:

程序设计及代码说明:

1. 无参构造器 SudokuMain（）的修改:

（1）创建一个用于选择难度的面板:

①创建 JPanel 对象 difficultyPanel;

②创建 JLabel 对象 difficultyLabel，传入参数“选择难度：”作为界面提示语;

③创建 JComboBox<String>对象 difficultyComboBox，并传入三个不同难度“Easy”、“Intermediate”、“Difficult”;

④创建 JButton 对象 startButton，命名为“开始游戏”，用户点击此按钮后即可开始数独游戏;

⑤分别将上述创建的 difficultyLabel、difficultyComboBox、startButton 对象通过 add（）方法传入 difficultyPanel 中。

（2）对 board 对象调用 setVisible（false）方法，将游戏面板隐藏，直至用户选择了难度。

（3）创建菜单栏:

①创建 JMenuBar 对象 menuBar，对其调用 setMenuBar（）方法；

②创建 JMenu 对象 fileMenu，命名为“File”，对 menuBar 调用 add（fileMenu）方法；

③分别创建三个 JMenuItem 对象 newGameItem、resetGameItem、exitItem，并分别对 fileMenu 调用 add 方法将上述三个 JMenuItem 对象传入，且可调用 addSeparator（）方法添加分割线，使界面更美观；

（4）为开始按钮添加动作监听器：

①对 startButton 调用 addActionListener 方法；

②重写 actionPerformed 方法：创建整数类型变量 difficultyLevel，其值为 difficultyComboBox 调用 getSelectedIndex（）方法再+1 的结果；对游戏面板 board 调用 newGame（difficultyLevel）方法，根据用户选择的难度创建一个新的数独游戏；对 difficultyPanel 调用 setVisible（false）方法，将选择难度的界面设为不可见模式；对 board 调用 setVisible（true）方法，将游戏面板设为可见；调用一次 pack（）方法；

（5）为 File 菜单添加动作监听器：

①对 newGameItem 调用 addActionListener 方法，并重写 actionPerformed 方法（仅调用 NewGame（）方法即完成对 actionPerformed 方法的重写）

②对 resetGameItem 和 exitItem 的操作如上，仅在重写 actionPerformed 方法时将对 NewGame（）的调用分别改为 ResetGame（）和 Exit（）的调用。

2. 编写 NewGame（）方法:

①设定一个整数类型变量 difficultyLevel，调用 JOptionPanel 中的 showOptionDialog 方法（具体参数可看代码展示部分），并将最后的值赋给 difficultyLevel;

②对 board 调用 newGame（difficultyLevel+1）方法，初始化一个数独游戏;

③调用 setVisible（true）方法使 board 面板可见;

④调用一次 pack（）方法。

3. 编写 ResetGame（）方法:

对 board 调用 resetGame（）方法，将游戏面板初始化。

4. 编写 Exit（）方法:

调用 System.exit（0）退出程序。

代码展示:

```
public class SudokuMain extends JFrame {
    no usages
    private static final long serialVersionUID = 1L;
    7 usages
    private GameBoardPanel board = new GameBoardPanel();

    1 usage
    public SudokuMain() {
        Container cp = getContentPane();
        cp.setLayout(new BorderLayout());

        //创建一个用于选择难度的面板
        JPanel difficultyPanel = new JPanel();
        JLabel difficultyLabel = new JLabel( text: "选择难度：");
        JComboBox<String> difficultyComboBox = new JComboBox<>(new String[]{"Easy", "Intermediate", "Difficult"});
        JButton startButton = new JButton( text: "开始游戏");
        difficultyPanel.add(difficultyLabel);
        difficultyPanel.add(difficultyComboBox);
        difficultyPanel.add(startButton);

        cp.add(difficultyPanel, BorderLayout.NORTH);
        cp.add(board, BorderLayout.CENTER);

        //最初将游戏面板隐藏，直到用户选择了难度
        board.setVisible(false);

        //创建菜单栏
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);

        //创建“File”菜单
        JMenu fileMenu = new JMenu( s: "File");
        menuBar.add(fileMenu);

        //创建菜单项并添加到“File”菜单
        JMenuItem newGameItem = new JMenuItem( text: "New Game");
        JMenuItem resetGameItem = new JMenuItem( text: "Reset Game");
        JMenuItem exitItem = new JMenuItem( text: "Exit");

        fileMenu.add(newGameItem);
        fileMenu.addSeparator(); //添加分隔线
        fileMenu.add(resetGameItem);
        fileMenu.addSeparator(); //添加分隔线
        fileMenu.add(exitItem);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setTitle("Sudoku");
        pack();

        //将窗口居中显示在屏幕上
        setLocationRelativeTo(null);
        setVisible(true);

        //为开始按钮添加动作监听器
        startButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //获取用户选择的难度级别
                int difficultyLevel = difficultyComboBox.getSelectedIndex()+1;

                //使用所选的难度级别初始化新游戏
                board.newGame(difficultyLevel);

                //隐藏难度选择面板，显示游戏面板
                difficultyPanel.setVisible(false);
                board.setVisible(true);
                pack();
            }
        });
```

```
        //为菜单项添加动作监听器
        newGameItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //处理NewGame操作
                NewGame();
            }
        });

        resetGameItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //处理ResetGame操作
                ResetGame();
            }
        });

        exitItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                //处理Exit操作
                Exit();
            }
        });
    }
```

```
    1 usage
    private void NewGame() {
        int difficultyLevel = JOptionPane.showOptionDialog( parentComponent: null,
                message: "选择难度级别", title: "选择难度", JOptionPane.DEFAULT_OPTION,
                JOptionPane.INFORMATION_MESSAGE, icon: null,
                new Object[]{ "Easy", "Intermediate", "Difficult"}, initialValue: "Easy");

        board.newGame( degree: difficultyLevel+1);
        board.setVisible(true);
        pack();
    }

    1 usage
    private void ResetGame() {
        board.resetGame();
    }

    1 usage
    private void Exit() {
        //直接退出程序
        System.exit( status: 0);
    }
```

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() { new SudokuMain(); }
        });
    }
}
```

新添部分

一、NumberGenerator 类：

程序设计及代码说明：

1. 编写 isValidMove（int[][] sudoku, int row, int col, int num）方法，检查给定位置放置 num 数字是否符合数独规则：

（1）调用 for 循环，对 sudoku 在 row 行的每个已填充的数字和在 col 列的每个已填充的数字进行检索，只要存在与 num 相同的数字，则返回 false；

（2）定义整数类型变量 startRow 和 startCol，其值分别为 row-row%3 和 col-col%3，使两者的值仅可能为 0,3,6；

（3）调用两层 for 循环，对 startRow 开始的 3*3 大小的数组进行遍历，只要存在与 num 相同的数字，则返回 false；

（4）若 num 在上述检索完成后，判定其符合数独的规则，返回 true。

2. 编写 boolean 类型 addNumber（int[][] sudoku, int row, int col）方法，用于生成独立数组：

（1）用 if 语句判断 row 是否为 9，若是则返回 true；

（2）创建 List 类对象 numbers，其包含 1 到 9 之间的所有整数，并调用 Collections.shuffle（numbers）方法将 numbers 中的元素打乱；

（3）调用 for 循环依次检索 numbers 中的元素，并将当前检索到的值令为 num；用 if 语句调用 isValidMove（sudoku，row，col，

num），若返回 true 则令 sudoku[row][col]的值为 num，并定义当前行的下一行位置为 nextRow，在 nextRow=col=8 的时候将 row 的值+1，定义当前列的下一列位置为 nextCol，其值为（col+1）%9 的结果；用 if 语句判断 addNumber（sudoku，nextRow，nextCol）方法为 true 时，返回 true，否则将 sudoku 在 row 行 col 列的值置为 0 进行回溯；

（4）若上述 for 循环填充失败，则返回 false。

3. 编写 int[][]类型 getNumbers（）方法：

定义 9*9 大小的二维数组 numbers，调用 addNumber（numbers，0,0）方法，最后返回 numbers。

代码展示：

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

1 usage
public class NumberGenerator {
    //生成一个符合数独游戏规则的 9x9 数组
    1 usage
    public static int[][] getNumbers() {
        int[][] numbers = new int[9][9];
        addNumber(numbers, row: 0, col: 0);
        return numbers;
    }

    //递归，用于生成数独数组
    2 usages
    private static boolean addNumber(int[][] sudoku, int row, int col) {
        if (row == 9) {
            return true;
        }

        //打乱1到9的数字顺序
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);
        Collections.shuffle(numbers);
```

```
        //尝试填充当前位置
        for (int num : numbers) {
            if (isValidMove(sudoku, row, col, num)) {
                sudoku[row][col] = num;

                int nextRow = col == 8 ? row + 1 : row;
                int nextCol = (col + 1) % 9;

                if (addNumber(sudoku, nextRow, nextCol)) {
                    return true;
                }
                //如果当前放置不符合规则，则回溯
                sudoku[row][col] = 0;
            }
        }
        return false;
    }
    //检查在给定位置放置数字是否符合数独规则
    1 usage
    private static boolean isValidMove(int[][] sudoku, int row, int col, int num) {
        //检查'num'是否在同一行和同一列中
        for (int i = 0; i < 9; i++) {
            if (sudoku[row][i] == num || sudoku[i][col] == num) {
                return false;
            }
        }

        //检查'num'是否在同一3*3子网格中
        int startRow = row - row % 3;
        int startCol = col - col % 3;
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (sudoku[startRow + i][startCol + j] == num) {
                    return false;
                }
            }
        }
        return true;
    }
}
```

二、BooleanGenerator 类：

程序设计及代码说明：

1. 编写 change（boolean[][] array，int number）方法：

（1）创建 Random 对象 random；

（2）定义 needChange，令其等于 number*9+9 的结果，needChange 的值即为将要更改为 false 的单元格数；

（3）调用 for 循环，次数为 needChange 次，定义两个整数类型变量 row 和 col，其值均使用 random.nextInt（9）进行计算，调用 if 语句判断 array[row][col]位置是否为 false，若是则此次循环无效，令循环次数+1，若 array[row][col]为 true，则将其改为 false。

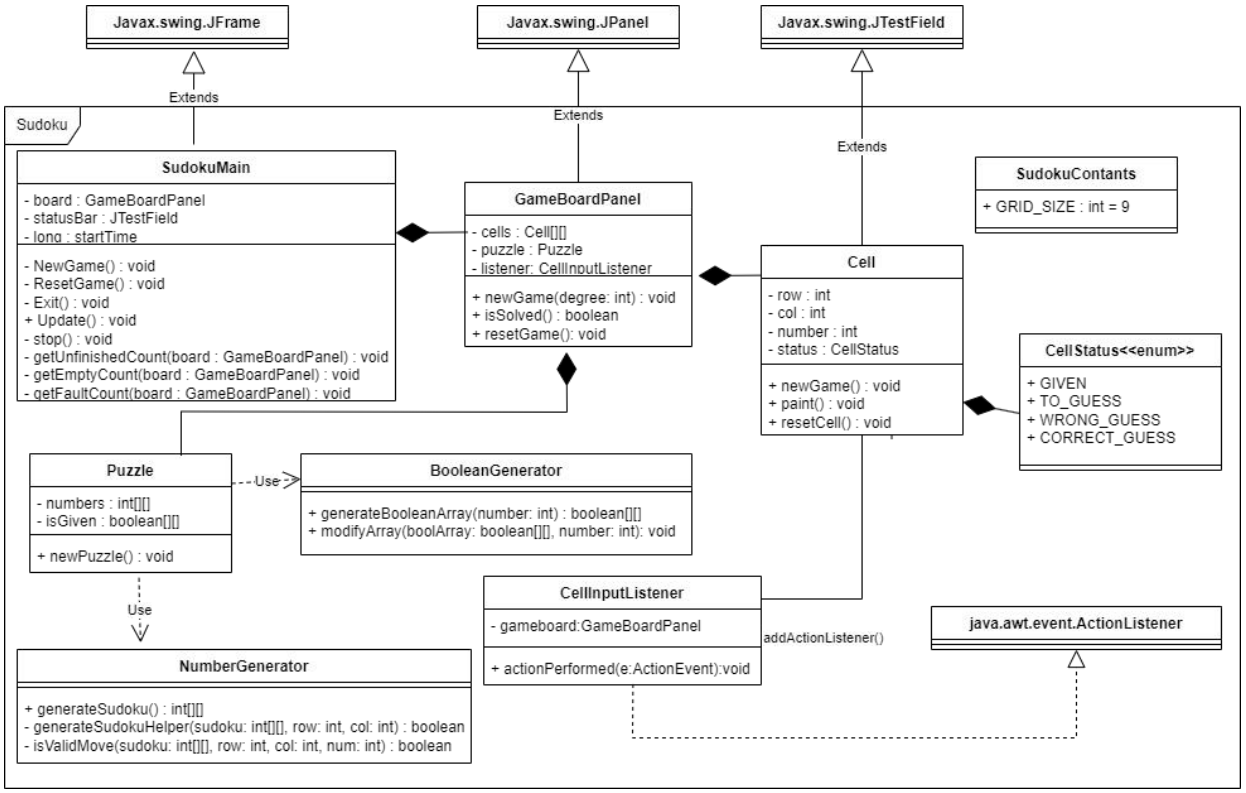
代码展示：

```
import java.util.Arrays;
import java.util.Random;

1 usage
public class BooleanGenerator {
    //生成一个 9x9 的布尔类型二维数组
    1 usage
    public static boolean[][] getBoolean(int number) {
        boolean[][] array = new boolean[9][9];
        for (int i = 0; i < 9; i++) {
            Arrays.fill(array[i], val: true);
        }
        change(array,number);
        return array;
    }

    //根据输入的数字随机更改数组中的元素为 false
    1 usage
    public static void change(boolean[][] array, int number) {
        Random random = new Random();
```

```
        // 根据输入的数字确定要更改的元素数量
        int needChange = (number+1) * 9;

        // 随机更改相应数量的元素为 false
        for (int i = 0; i < needChange; i++) {
            int row = random.nextInt( bound: 9);
            int col = random.nextInt( bound: 9);
            if(array[row][col] == false)
                i--;
            array[row][col] = false;
        }
    }
}
```

扩展部分

扩展部分 UMI 图:



一、SudokuMain 类:

程序设计及代码说明:

1. 添加两个私有成员，分别是 JTestField 类的对象 statusBar 和 long 对象 startTime, statusBar 用于给主界面添加状态栏, startTime 用于记录游戏开始的时间；‘’

2. 添加 stop（）方法，用于暂停状态栏的时间：编写 if 语句判断 timer 不为空且正在运行时，对 timer 调用 stop（）方法；

3. 添加 getTime（）方法，用于记录游戏时间和更新状态栏:

（1）对私有成员 startTime 调用 System.currentTimeMillis（）方法，记录用户点击开始游戏时的时间；

（2）定义 Timer 类对象 timer，并输入参数 1000 使其每秒更新一次。重写 actionPerformed 方法，仅书写 Update（board）即可；对 timer 调用 start（）方法启动计时器。

4. 添加 getUnfinishedCount（GameBoardPanel board）方法：

（1）定义临时整数类型变量 unfinishedCount，初始值为 0，用于记录棋盘中未完成的单元格的数量；

（2）调用两次 for 循环，依次检索 9*9 大小的棋盘中的每一个单元格的状态，判断检索到的单元格的状态（status）是否为 TO_GUESS 或者 WRONG_GUESS，若是则 unfinishedCount 的值加一；

（3）检索结束后返回 unfinishedCount 的值；

5. 同理编写 getEmptyCount（GameBoardPanel board）方法，用于获取未填写的单元格的数量；

6. 同理编写 getFaultCount（GameBoardPanel board）方法，用于获取填写错误的单元格的数量；

7. 添加方法 Update（GameBoardPanel board），用于更新状态栏：

（1）定义三个临时变量 unfinishedCount、nodoCount、faultCount，分别将 getUnfinishedCount、getEmptyCount、getFaultCount 方法返回的值赋给对应的临时变量；

（3）定义两个 long 类型变量 currentTime 和 elapsedTime，分别记录当前时间和进行游戏的时间，其中 currentTime 的值直接调用 System.currentTimeMillis （）即可，elapsedTime 的值为 currentTime 的值减去 startTime 的值的结果;

（4）对状态栏 statusBar 调用 setTest 方法，并输入需要显示的内容;

（5）用 if 语句判断 unfinishedCount 是否为 0，为 0 时棋盘上所有待完成的单元格均已被正确填充，判定此时游戏结束，调用 stop（）方法停止对游戏时间的更新。

8. 修改无参构造器 SudokuMain（）：

（1）将棋盘 board 加入 Container 对象 cp 后，创建状态栏 statusBar，并对其调用 setEditable（false）方法将其设定为不可编辑状态，最后对 cp 调用 add 方法将 statusBar 加入游戏主界面并书写 BorderLayout.SOUTH 后即可将状态栏添加至游戏界面的底端。

（2）调用 getTime（）方法，在开始按钮的动作添加器中增添记录时间功能和更新状态栏功能。

9. 修改 NewGame（）方法和 ResetGame（）方法：分别在方法体最后添加 getTime（）方法，使用户在菜单栏选择开始新游戏或者重置游戏后记录时间的数据置零。

代码展示：

1.添加私有成员：

```
    4 usages
    private static JTextField statusBar; //状态栏
    2 usages
    private long startTime; //游戏开始时间
```

2.在无参构造器内添加状态栏:

```
//创建状态栏并添加到窗口底部
statusBar = new JTextField();
statusBar.setEditable(false); // 让它不可编辑
cp.add(statusBar, BorderLayout.SOUTH);
```

3.添加的新方法:

```
//更新状态栏
2 usages
public static void Update(GameBoardPanel board) {
    SudokuMain sudokuMain = (SudokuMain) board.getTopLevelAncestor();
    if (sudokuMain != null) {
        int unfinishedCount = getUnfinishedCount(board);
        int nodoCount = getEmptyCount(board);
        int faultCount = getFaultCount(board);

        //计算已经过的时间（秒）
        long currentTime = System.currentTimeMillis();
        long elapsedTime = (currentTime - sudokuMain.startTime) / 1000;

        statusBar.setText("未完成的单元格: " + unfinishedCount +
                "   未填写的单元格: " + nodoCount +
                "   填写错误的单元格: " + faultCount +
                "   游戏时间: " + elapsedTime + " 秒");

        //判断游戏是否完成，如果完成则停止计时器
        if (unfinishedCount == 0) {
            sudokuMain.stop();
        }
    }
}

//暂停时间
1 usage
private void stop() {
    if (timer != null && timer.isRunning()) {
        timer.stop();
    }
}
```

```
//获取游戏时间并更新状态栏
3 usages
private void getTime(){
    //记录游戏开始时间
    startTime = System.currentTimeMillis();

    //启动计时器，每秒更新状态栏
    timer = new Timer( delay: 1000, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent actionEvent) {
            Update(board);
        }
    });
    timer.start();
}

//获取未完成的单元格数量
1 usage
private static int getUnfinishedCount(GameBoardPanel board) {
    int unfinishedCount = 0;
    for (int row = 0; row < SudokuConstants.GRID_SIZE; row++) {
        for (int col = 0; col < SudokuConstants.GRID_SIZE; col++) {
            if (board.cells[row][col].status == CellStatus.TO_GUESS || board.cells[row][col].status == CellStatus.WRONG_GUESS) {
                unfinishedCount++;
            }
        }
    }
    return unfinishedCount;
}
```

```
//获取未填写的单元格数量
1 usage
private static int getEmptyCount(GameBoardPanel board) {
    int nodoCount = 0;
    for (int row = 0; row < SudokuConstants.GRID_SIZE; row++) {
        for (int col = 0; col < SudokuConstants.GRID_SIZE; col++) {
            if (board.cells[row][col].status == CellStatus.TO_GUESS) {
                nodoCount++;
            }
        }
    }
    return nodoCount;
}

//获取填写错误的单元格数量
1 usage
private static int getFaultCount(GameBoardPanel board) {
    int faultCount = 0;
    for (int row = 0; row < SudokuConstants.GRID_SIZE; row++) {
        for (int col = 0; col < SudokuConstants.GRID_SIZE; col++) {
            if (board.cells[row][col].status == CellStatus.WRONG_GUESS) {
                faultCount++;
            }
        }
    }
    return faultCount;
}
```

二、Cell 类:

由于对 Cell 类的更改仅涉及对单元格的背景颜色、字体的更改，并无技术含量，故直接展示改动部分代码:

1. 成员对象的改动:

```
// 预先定义好在单元格不同状态下的颜色常量
2 usages
public static final Color BG_GIVEN = new Color( r: 234,  g: 180,  b: 109); // RGB
1 usage
public static final Color BG_FAULT_GIVEN = new Color( r: 145,  g: 70,  b: 69); // RGB
1 usage
public static final Color FG_GIVEN = Color.BLACK;
2 usages
public static final Color FG_NOT_GIVEN = Color.GRAY;
1 usage
public static final Color FG_CT_GIVEN = new Color( r: 48,  g: 166,  b: 22);
1 usage
public static final Color FG_FT_GIVEN = Color.RED;
2 usages
public static final Color BG_TO_GUESS  = new Color( r: 250,  g: 218,  b: 185); // RGB
1 usage
public static final Font FONT_NUMBERS = new Font( name: "OCR A Extended", Font.PLAIN,  size: 28);
```

2. Paint（）方法的改动:

```
/** 单元格的外观取决于状态 */
2 usages
public void paint() {
    // 以下方法的调用都使用了super，其实是没有必要的，主要是让同学们了解这些方法都是JTextField类中定义的方法
    if (status == CellStatus.GIVEN) {
        super.setText(number + "");
        super.setEditable(false);
        super.setBackground(BG_GIVEN);
        super.setForeground(FG_GIVEN);
    } else if (status == CellStatus.TO_GUESS) {
        super.setText("");
        super.setEditable(true);
        super.setBackground(BG_TO_GUESS);
        super.setForeground(FG_NOT_GIVEN);
    } else if (status == CellStatus.CORRECT_GUESS) {
        super.setBackground(BG_GIVEN);
        super.setForeground(FG_CT_GIVEN);
        super.setEditable(false);
    } else if (status == CellStatus.WRONG_GUESS) {
        super.setBackground(BG_FAULT_GIVEN);
        super.setForeground(FG_FT_GIVEN);
    }
}
```

分值表

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	7
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。	6
填表人： 王兴昊		