**Sudoku Class Diagram**

Diagram

Description automatically generated

**Sudoku Pseudocode:**

**Button Controller Class:**

**//Class controls the user actions from the ButtonPanel**

public class ButtonController implements ActionListener {

private Game game;

**// Constructor, sets game. @param game Game to be set.**

public ButtonController(Game game) {

this.game = game;

}

**// Performs action after user pressed button. @param e ActionEvent.**

public void actionPerformed(ActionEvent e) {

if (e.getActionCommand().equals("New"))

game.newGame();

else if (e.getActionCommand().equals("Check"))

game.checkGame();

else if (e.getActionCommand().equals("Exit"))

System.exit(0);

else if (e.getActionCommand().equals("Help on"))

game.setHelp(((JCheckBox)e.getSource()).isSelected());

else

game.setSelectedNumber(Integer.parseInt(e.getActionCommand()));

}

}

**Sudoku Controller Class:**

**//Class controls the user actions from SudokuPanel.**

public class SudokuController implements MouseListener {

private SudokuPanel sudokuPanel; **// Panel to control.**

private Game game; **// Current Sudoku game.**

**//Constructor, sets game. @param game Game to be set.**

public SudokuController(SudokuPanel sudokuPanel, Game game) {

this.sudokuPanel = sudokuPanel;

this.game = game;

}

**//Recovers if user clicked field in game. If so it sets the selected number at clicked position in game and updates clicked field. If user clicked a field and used left mouse button, number at clicked position will be cleared in game and clicked field will be updated. @param e MouseEvent.**

public void mousePressed(MouseEvent e) {

JPanel panel = (JPanel)e.getSource();

Component component = panel.getComponentAt(e.getPoint());

if (component instanceof Field) {

Field field = (Field)component;

int x = field.getFieldX();

int y = field.getFieldY();

if (e.getButton() == MouseEvent.BUTTON1 && (game.getNumber(x, y) == 0 || field.getForeground().equals(Color.BLUE))) {

int number = game.getSelectedNumber();

if (number == -1)

return;

game.setNumber(x, y, number);

field.setNumber(number, true);

} else if (e.getButton() == MouseEvent.BUTTON3 && !field.getForeground().equals(Color.BLACK)) {

game.setNumber(x, y, 0);

field.setNumber(0, false);

}

sudokuPanel.update(game, UpdateAction.CANDIDATES);

}

}

public void mouseClicked(MouseEvent e) { }

public void mouseEntered(MouseEvent e) { }

public void mouseExited(MouseEvent e) { }

public void mouseReleased(MouseEvent e) { }

}

**Sudoku Game Class:**

**//Class represents a Sudoku game. It contains the solution, the user input, the selected number and methods to check the validation of the user input.**

public class Game extends Observable {

private int[][] solution; **// Generated solution.**

private int[][] game; **// Generated game with user input.**

private boolean[][] check; **// Holder for checking validity of game.**

private int selectedNumber; **// Selected number by user.**

private boolean help; **// Help turned on or off.**

**// Constructor**

public Game() {

newGame();

check = new boolean[9][9];

help = true;

}

**//Generates a new Sudoku game. All observers will be notified, update action: new game.**

public void newGame() {

solution = generateSolution(new int[9][9], 0);

game = generateGame(copy(solution));

setChanged();

notifyObservers(UpdateAction.NEW\_GAME);

}

**//Checks user input again, the solution and puts it into a check matrix. All observers will be notified, update action: check.**

public void checkGame() {

selectedNumber = 0;

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++)

check[y][x] = game[y][x] == solution[y][x];

}

setChanged();

notifyObservers(UpdateAction.CHECK);

}

**//Sets help turned on or off. All observers will be notified, update action: help. @param help True for help on, false for help off.**

public void setHelp(boolean help) {

this.help = help;

setChanged();

notifyObservers(UpdateAction.HELP);

}

**// Sets selected number to user input. All observers will be notified, update action: selected number. @param selectedNumber Number selected by user.**

public void setSelectedNumber(int selectedNumber) {

this.selectedNumber = selectedNumber;

setChanged();

notifyObservers(UpdateAction.SELECTED\_NUMBER);

}

**//Returns number selected user. @return Number selected by user.**

public int getSelectedNumber() {

return selectedNumber;

}

**//Returns whether help is turned on or off. @return True if help is turned on, false if help is turned off.**

public boolean isHelp() {

return help;

}

**//Returns whether selected number is candidate at given position. @param x X position in game. @param y Y position in game. @return True if selected number on given position is candidate, false otherwise.**

public boolean isSelectedNumberCandidate(int x, int y) {

return game[y][x] == 0 && isPossibleX(game, y, selectedNumber)

&& isPossibleY(game, x, selectedNumber) && isPossibleBlock(game, x, y, selectedNumber);

}

**//Sets given number on given position in the game. @param x, The x position in the game. @param y, The y position in the game. @param number, The number to be set.**

public void setNumber(int x, int y, int number) {

game[y][x] = number;

}

**//Returns number of given position. @param x, X position in game. @param y, Y position in game. @return, Number of given position.**

public int getNumber(int x, int y) {

return game[y][x];

}

**//Returns whether user input is valid of given position. @param x, X position in game. @param y, Y position in game. @return, True if user input of given position is valid, false otherwise.**

public boolean isCheckValid(int x, int y) {

return check[y][x];

}

**//Returns whether given number is candidate on x axis for given game. @param game, Game to check. @param y, Position of x axis to check. @param number, Number to check. @return, True if number is candidate on x axis, false otherwise.**

private boolean isPossibleX(int[][] game, int y, int number) {

for (int x = 0; x < 9; x++) {

if (game[y][x] == number)

return false;

}

return true;

}

**//Returns whether given number is candidate on y axis for given game. @param game, Game to check. @param x, Position of y axis to check. @param number, Number to check. @return, True if number is candidate on y axis, false otherwise.**

private boolean isPossibleY(int[][] game, int x, int number) {

for (int y = 0; y < 9; y++) {

if (game[y][x] == number)

return false;

}

return true;

}

**//Returns whether given number is candidate in block for given game. @param game, Game to check. @param x, Position of number on x axis in game to check. @param y, Position of number on y axis in game to check. @param number, Number to check. @return, True if number is candidate in block, false otherwise.**

private boolean isPossibleBlock(int[][] game, int x, int y, int number) {

int x1 = x < 3 ? 0 : x < 6 ? 3 : 6;

int y1 = y < 3 ? 0 : y < 6 ? 3 : 6;

for (int yy = y1; yy < y1 + 3; yy++) {

for (int xx = x1; xx < x1 + 3; xx++) {

if (game[yy][xx] == number)

return false;

}

}

return true;

}

**//Returns next posible number from list for given position or -1 when list is empty. @param game, Game to check. @param x, X position in game. @param y, Y position in game. @param numbers, List of remaining numbers. @return, Next possible number for position in game or -1 when list is empty.**

private int getNextPossibleNumber(int[][] game, int x, int y, List<Integer> numbers) {

while (numbers.size() > 0) {

int number = numbers.remove(0);

if (isPossibleX(game, y, number) && isPossibleY(game, x, number) && isPossibleBlock(game, x, y, number))

return number;

}

return -1;

}

**// Generates Sudoku game solution. @param game, Game to fill, user should pass 'new int[9][9]'. @param index, Current index, user should pass 0. @return, Sudoku game solution.**

private int[][] generateSolution(int[][] game, int index) {

if (index > 80)

return game;

int x = index % 9;

int y = index / 9;

List<Integer> numbers = new ArrayList<Integer>();

for (int i = 1; i <= 9; i++) numbers.add(i);

Collections.shuffle(numbers);

while (numbers.size() > 0) {

int number = getNextPossibleNumber(game, x, y, numbers);

if (number == -1)

return null;

game[y][x] = number;

int[][] tmpGame = generateSolution(game, index + 1);

if (tmpGame != null)

return tmpGame;

game[y][x] = 0;

}

return null;

}

**//Generates Sudoku game from solution. @param game, Game to be generated, user should pass a solution. @return, Generated Sudoku game.**

private int[][] generateGame(int[][] game) {

List<Integer> positions = new ArrayList<Integer>();

for (int i = 0; i < 81; i++)

positions.add(i);

Collections.shuffle(positions);

return generateGame(game, positions);

}

**//Generates Sudoku game from solution, user should use the other generateGame method. This method simple removes a number at a position. If the game isn't valid after this action, the game will be brought back to previous state. @param game,Game to be generated. @param positions, List of remaining positions to clear. @return, Generated Sudoku game.**

private int[][] generateGame(int[][] game, List<Integer> positions) {

while (positions.size() > 0) {

int position = positions.remove(0);

int x = position % 9;

int y = position / 9;

int temp = game[y][x];

game[y][x] = 0;

if (!isValid(game))

game[y][x] = temp;

}

return game;

}

**// Checks whether given game is valid. @param game, Game to check. @return, True if game is valid, false otherwise.**

private boolean isValid(int[][] game) {

return isValid(game, 0, new int[] { 0 });

}

**//Checks whether given game is valid, user should use the other isValid method. There may only be one solution @param game, Game to check. @param index, Current index to check. @param numberOfSolutions, Number of found solutions. Int[] instead of, int because of pass by reference @return, True if game is valid, false otherwise.**

private boolean isValid(int[][] game, int index, int[] numberOfSolutions) {

if (index > 80)

return ++numberOfSolutions[0] == 1;

int x = index % 9;

int y = index / 9;

if (game[y][x] == 0) {

List<Integer> numbers = new ArrayList<Integer>();

for (int i = 1; i <= 9; i++)

numbers.add(i);

while (numbers.size() > 0) {

int number = getNextPossibleNumber(game, x, y, numbers);

if (number == -1)

break;

game[y][x] = number;

if (!isValid(game, index + 1, numberOfSolutions)) {

game[y][x] = 0;

return false;

}

game[y][x] = 0;

}

} else if (!isValid(game, index + 1, numberOfSolutions))

return false;

return true;

}

**//Copies a game. @param game, Game to be copied. @return, Copy of given game.**

private int[][] copy(int[][] game) {

int[][] copy = new int[9][9];

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++)

copy[y][x] = game[y][x];

}

return copy;

}

**/\***

**\* Prints given game to console. Used for debug.**

**\***

**\* @param game Game to be printed.**

**\***

**private void print(int[][] game) {**

**System.out.println();**

**for (int y = 0; y < 9; y++) {**

**for (int x = 0; x < 9; x++)**

**System.out.print(" " + game[y][x]);**

**System.out.println();**

**}**

**}\*/**

}

**Update Action:**

**// Enumeration used to inform observers what to update.**

public enum UpdateAction {

NEW\_GAME,

CHECK,

SELECTED\_NUMBER,

CANDIDATES,

HELP

}

**Button Panel Class:**

**//Class draws the button panel and reacts to updates from the model.**

public class ButtonPanel extends JPanel implements Observer {

JButton btnNew, btnCheck, btnExit; **// Used buttons.**

JCheckBox cbHelp; **// Used check box.**

ButtonGroup bgNumbers; **// Group for grouping the toggle buttons.**

JToggleButton[] btnNumbers; **// Used toggle buttons.**

**//Constructs the panel and arranges all components.**

public ButtonPanel() {

super(new BorderLayout());

JPanel pnlAlign = new JPanel();

pnlAlign.setLayout(new BoxLayout(pnlAlign, BoxLayout.PAGE\_AXIS));

add(pnlAlign, BorderLayout.NORTH);

JPanel pnlOptions = new JPanel(new FlowLayout(FlowLayout.LEADING));

pnlOptions.setBorder(BorderFactory.createTitledBorder(" Options "));

pnlAlign.add(pnlOptions);

btnNew = new JButton("New");

btnNew.setFocusable(false);

pnlOptions.add(btnNew);

btnCheck = new JButton("Check");

btnCheck.setFocusable(false);

pnlOptions.add(btnCheck);

btnExit = new JButton("Exit");

btnExit.setFocusable(false);

pnlOptions.add(btnExit);

JPanel pnlNumbers = new JPanel();

pnlNumbers.setLayout(new BoxLayout(pnlNumbers, BoxLayout.PAGE\_AXIS));

pnlNumbers.setBorder(BorderFactory.createTitledBorder(" Numbers "));

pnlAlign.add(pnlNumbers);

JPanel pnlNumbersHelp = new JPanel(new FlowLayout(FlowLayout.LEADING));

pnlNumbers.add(pnlNumbersHelp);

cbHelp = new JCheckBox("Help on", true);

cbHelp.setFocusable(false);

pnlNumbersHelp.add(cbHelp);

JPanel pnlNumbersNumbers = new JPanel(new FlowLayout(FlowLayout.LEADING));

pnlNumbers.add(pnlNumbersNumbers);

bgNumbers = new ButtonGroup();

btnNumbers = new JToggleButton[9];

for (int i = 0; i < 9; i++) {

btnNumbers[i] = new JToggleButton("" + (i + 1));

btnNumbers[i].setPreferredSize(new Dimension(40, 40));

btnNumbers[i].setFocusable(false);

bgNumbers.add(btnNumbers[i]);

pnlNumbersNumbers.add(btnNumbers[i]);

}

}

**// Method called when model sends update notification. @param o, The model. @param arg, The UpdateAction.**

public void update(Observable o, Object arg) {

switch ((UpdateAction)arg) {

case NEW\_GAME:

case CHECK:

bgNumbers.clearSelection();

break;

}

}

**//Adds controller to all components. @param buttonController, Controller which controls all user actions.**

public void setController(ButtonController buttonController) {

btnNew.addActionListener(buttonController);

btnCheck.addActionListener(buttonController);

btnExit.addActionListener(buttonController);

cbHelp.addActionListener(buttonController);

for (int i = 0; i < 9; i++)

btnNumbers[i].addActionListener(buttonController);

}

}

**Field Class:**

**//Class represents a field on the SudokuPanel.**

public class Field extends JLabel {

private int x; // X position in game.

private int y; // Y position in game.

**//Constructs the label and sets x and y positions in game. @param x, X position in game. @param y, Y position in game.**

public Field(int x, int y) {

super("", CENTER);

this.x = x;

this.y = y;

setPreferredSize(new Dimension(40, 40));

setBorder(BorderFactory.createLineBorder(Color.GRAY));

setFont(new Font(Font.DIALOG, Font.PLAIN, 20));

setOpaque(true);

}

**//Sets number and foreground color according to userInput. @param number, Number to be set. @param userInput, Boolean indicating number is user input or not.**

public void setNumber(int number, boolean userInput) {

setForeground(userInput ? Color.BLUE : Color.BLACK);

setText(number > 0 ? number + "" : "");

}

**// Returns x position in game. @return X position in game.**

public int getFieldX() {

return x;

}

**// Return y position in game. @return Y position in game.**

public int getFieldY() {

return y;

}

}

**Sudoku Class:**

**// Main class of program.**

public class Sudoku extends JFrame {

public Sudoku() {

super("Sudoku");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

getContentPane().setLayout(new BorderLayout());

Game game = new Game();

ButtonController buttonController = new ButtonController(game);

ButtonPanel buttonPanel = new ButtonPanel();

buttonPanel.setController(buttonController);

add(buttonPanel, BorderLayout.EAST);

SudokuPanel sudokuPanel = new SudokuPanel();

SudokuController sudokuController = new SudokuController(sudokuPanel, game);

sudokuPanel.setGame(game);

sudokuPanel.setController(sudokuController);

add(sudokuPanel, BorderLayout.CENTER);

game.addObserver(buttonPanel);

game.addObserver(sudokuPanel);

pack();

setLocationRelativeTo(null);

setVisible(true);

}

**// Main entry point of program. @param args Command line arguments.**

public static void main(String[] args) {

// Use System Look and Feel

try { UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName()); }

catch (Exception ex) { ex.printStackTrace(); }

new Sudoku();

}

}

**Sudoku Panel Class:**

**//Class draws the sudoku panel and reacts to updates from the model.**

public class SudokuPanel extends JPanel implements Observer {

**// Color constant for candidates.**

private static final Color COLOR\_CANDIDATE = new Color(102, 153, 255);

private Field[][] fields; **// Array of fields.**

private JPanel[][] panels; **// Panels holding the fields.**

**//Constructs the panel, adds sub panels and adds fields to these sub panels.**

public SudokuPanel() {

super(new GridLayout(3, 3));

panels = new JPanel[3][3];

for (int y = 0; y < 3; y++) {

for (int x = 0; x < 3; x++) {

panels[y][x] = new JPanel(new GridLayout(3, 3));

panels[y][x].setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY));

add(panels[y][x]);

}

}

fields = new Field[9][9];

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x] = new Field(x, y);

panels[y / 3][x / 3].add(fields[y][x]);

}

}

}

**//Method called when model sends update notification. @param o, The model. @param arg, The UpdateAction.**

public void update(Observable o, Object arg) {

switch ((UpdateAction)arg) {

case NEW\_GAME:

setGame((Game)o);

break;

case CHECK:

setGameCheck((Game)o);

break;

case SELECTED\_NUMBER:

case CANDIDATES:

case HELP:

setCandidates((Game)o);

break;

}

}

**//Sets the fields corresponding to given game. @param game, Game to be set.**

public void setGame(Game game) {

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x].setBackground(Color.WHITE);

fields[y][x].setNumber(game.getNumber(x, y), false);

}

}

}

**// Sets fields validity according to given game. @param game, Current game.**

private void setGameCheck(Game game) {

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x].setBackground(Color.WHITE);

if (fields[y][x].getForeground().equals(Color.BLUE))

fields[y][x].setBackground(game.isCheckValid(x, y) ? Color.GREEN : Color.RED);

}

}

}

**//Shows the candidates according to given game. @param game, Current game.**

private void setCandidates(Game game) {

for (int y = 0; y < 9; y++) {

for (int x = 0; x < 9; x++) {

fields[y][x].setBackground(Color.WHITE);

if (game.isHelp() && game.isSelectedNumberCandidate(x, y))

fields[y][x].setBackground(COLOR\_CANDIDATE);

}

}

}

**//Adds controller to all sub panels. @param sudokuController, Controller which controls all user actions.**

public void setController(SudokuController sudokuController) {

for (int y = 0; y < 3; y++) {

for (int x = 0; x < 3; x++)

panels[y][x].addMouseListener(sudokuController);

}

}

}