```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class Main extends JFrame {

    public static void main(String[] args) {

        sudokuInterface sInterface = new sudokuInterface();        //calling sudokuInterface class
    }
}
class sudokuInterface {
    static JFrame frame = new JFrame("Sudoku Solver");
    static String[][] array2d = new String[9][9];
    static int[][] sudokuBoard = new int[9][9];
    static int[][] reducedDomainArray = new int[81][9];
    static JButton[][] buttonGrid = new JButton[9][9];     //creating a 2D button array
    static JMenuBar menuBar = new JMenuBar();
    static JButton solve = new JButton(" Solve ! ");
    static JButton clear = new JButton(" Clear ");
    static JButton openSudoku = new JButton(" Open Sudoku file ");

    sudokuInterface(){      //sudokuInterface constructor

        solve.setFont(new Font("Arial",Font.PLAIN,20));
        solve.setBorder(BorderFactory.createLineBorder(Color.black));

        openSudoku.setFont(new Font("Arial",Font.PLAIN,20));
        openSudoku.setBorder(BorderFactory.createLineBorder(Color.black));

        clear.setFont(new Font("Arial",Font.PLAIN,20));
        clear.setBorder(BorderFactory.createLineBorder(Color.black));
        menuBar.setPreferredSize(new Dimension(35,35));

        for (int i = 0; i <9 ; i++) {       //array 2d is initially empty
            for (int j = 0; j < 9; j++) {
                array2d[i][j] = " ";
            }
        }

        frame.setLayout(new GridLayout(9,9));
        for (int i = 0; i< buttonGrid.length; i++){       //creating a 9x9 sudoku grid
            for (int j = 0; j< buttonGrid.length; j++){
                buttonGrid[i][j] = new JButton("");       //all buttons initially to 0
                buttonGrid[i][j].setFont(new Font("Arial", Font.BOLD, 35));
                buttonGrid[i][j].setBackground(Color.white);
```

```
            buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(1,1,1,1,Color.BLACK));
            frame.add(buttonGrid[i][j]);      //add buttons to the frame
        }
    }

    selectedColors();      //assign colors to alternate to 3x3 squares
    assignBorders();       //bordered buttons to differentiate between 3x3 squares

    for (int i=0; i<9; i++){
        for (int j = 0; j< 9; j++) {
            int finalI = i;
            int finalJ = j;
            buttonGrid[i][j].addActionListener(new ActionListener() {      //action listener to all
buttons to input the puzzle
                @Override
                public void actionPerformed(ActionEvent actionEvent) {
                    try {
                        //taking input from the user
                        String numberInput = JOptionPane.showInputDialog(null,"enter number from 1
to 9");
                        int numberInputInt = Integer.parseInt(numberInput);

                        //input should lie between 1 and 9 only
                        while (!(numberInputInt>0 && numberInputInt <10)){
                            JOptionPane.showMessageDialog(null,"Please enter correct input");
                            numberInput = JOptionPane.showInputDialog(null,"enter number from 1 to
9");
                            numberInputInt = Integer.parseInt(numberInput);
                        }

                        //setting numbers to the button array and the 2D array
                        buttonGrid[finalI][finalJ].setText(numberInput);
                        array2d[finalI][finalJ] = numberInput;
                        buttonGrid[finalI][finalI].setForeground(Color.black);
                    }
                    catch (NumberFormatException e){
                        JOptionPane.showMessageDialog(null, "Please enter correct input");
                    }
                }
            });

            //adding mouse listener when we hover over the buttons
            buttonGrid[i][j].addMouseListener(new MouseListener() {
                @Override
                public void mouseClicked(MouseEvent mouseEvent) {
                }
                @Override
                public void mousePressed(MouseEvent mouseEvent) {
                }
                @Override
                public void mouseReleased(MouseEvent mouseEvent) {
                }
```

```
            @Override
            public void mouseEntered(MouseEvent mouseEvent) {
                buttonGrid[finalI][finalJ].setBackground(Color.yellow);
            }
            @Override
            public void mouseExited(MouseEvent mouseEvent) {     //change the color back to
normal
                selectedColors();
                buttonGrid[finalI][finalJ].setBackground(Color.white);
            }
        });
    }
}

//opening file input from the openSudoku button
openSudoku.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        JFileChooser fileChooser = new JFileChooser("c:");
        // Invoke the showsOpenDialog function to show the save dialog
        int r = fileChooser.showOpenDialog(null);

        // If the user selects a file
        if (r == JFileChooser.APPROVE_OPTION) {
            // Set the label to the path of the selected directory
            File file = new File(fileChooser.getSelectedFile().getAbsolutePath());

            try {

                String s = "", sudokuFile = "";
                FileReader fileReader = new FileReader(file);
                BufferedReader bufferedReader = new BufferedReader(fileReader);

                // Initialize sudokuFile
                sudokuFile = bufferedReader.readLine();

                // Take the input from the file
                while ((s = bufferedReader.readLine()) != null) {
                    sudokuFile = sudokuFile + "," + s;
                }
                inputSudoku(sudokuFile);     //pass the csv file to a function
            }
            catch (Exception evt) {
                JOptionPane.showMessageDialog(null, "Improper format, please try again.");
            }
        }
    }
});

solve.addActionListener(new ActionListener() {       //implementing the solve button
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
```

```
        try {
            for (int i = 0; i <9 ; i++) {
                for (int j = 0; j < 9; j++) {       //assigning the array2d array to an integer array

                    if (array2d[i][j].equals(" ")) sudokuBoard[i][j] = 0;
                    else {
                        sudokuBoard[i][j] = Integer.parseInt(array2d[i][j]);

                    }

                    //System.out.print(sudokuBoard[i][j] + " ");
                }
                System.out.println();
            }
            System.out.println();

            for (int i = 0; i <9 ; i++) {
                for (int j = 0; j <9 ; j++) {
                    if (sudokuBoard[i][j] != 0){
                        if (!checkConstraints(sudokuBoard, i, j, sudokuBoard[i][j])) {
                            JOptionPane.showMessageDialog(null,"Incorrect Input");
                            return;
                        }
                    }
                }
            }

            if (sudokuSolver(sudokuBoard))
            {
                showSolution(sudokuBoard);    //shows the sudoku solution on the GUI
            }

        }
        catch (NullPointerException e){
            JOptionPane.showMessageDialog(null, "input can't be empty !");
        }
    }
});

clear.addActionListener(new ActionListener() {      //clears the board
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        for (int i=0; i<9; i++){
            for (int j=0; j<9; j++){
                buttonGrid[i][j].setText("");
                sudokuBoard[i][j] = 0;
                array2d[i][j] = " ";
            }
        }
    }
});
```

```
    menuBar.add(solve);     //adding menubar and displaying the frame
    menuBar.add(openSudoku);
    menuBar.add(clear);
    frame.setJMenuBar(menuBar);
    frame.setVisible(true);
    frame.setSize(600,600);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null);
}

private void inputSudoku(String fileInput) {
    try {

        String[] sudokuInput = new String[81];

        //splitting the fileInput csv file and assigning it to an array
        sudokuInput = fileInput.split(",");

        for (int i = 0; i < sudokuInput.length; i++) {
            if (sudokuInput[i].equals("0")) sudokuInput[i] = " ";
        }

        //converting the 1D array to a 2D array
        int k=0;
        for(int i=0; i< 9; i++){
            if (k == (sudokuInput.length-1)) break;
            for(int j=0; j<9; j++){
                array2d[i][j] = sudokuInput[k];
                k++;
            }
        }

        //assigning that 2D array to the buttons array
        for (int i=0;i<9;i++){
            for (int j=0;j<9;j++){
                buttonGrid[i][j].setText(array2d[i][j]);
                buttonGrid[i][j].setForeground(Color.black);
            }
        }
    }
    catch (ArrayIndexOutOfBoundsException e){
        JOptionPane.showMessageDialog(null,"wrong input, please try again");
    }
}

//implementing the main solver function
private static boolean sudokuSolver(int[][] sudokuBoard)
{
    int row = -1;
    int col = -1;
    boolean checkIfEmpty = true;
    for (int i = 0; i < 9; i++)
```

```
    {
        for (int j = 0; j < 9; j++)
        {
            reducedDomain(sudokuBoard,i,j);
            //if the square is empty, try to fill the board
            if (sudokuBoard[i][j] == 0)
            {
                row = i;
                col = j;
                //if we still have some remaining missing values in the board
                checkIfEmpty = false;
                break;
            }
        }
        if (!checkIfEmpty)
        {
            break;
        }
    }

    //if no empty space is remaining, return true and exit the function
    if (checkIfEmpty)
    {
        return true;
    }

    // else for each row use recursive backtracking
    for (int number = 1; number <= 9; number++)
    {
        if (checkConstraints(sudokuBoard, row, col, number))
        {
            sudokuBoard[row][col] = number;
            if (sudokuSolver(sudokuBoard))
            {
                // print(board);
                showSolution(sudokuBoard);
                return true;
            }
            else
            {
                sudokuBoard[row][col] = 0; // replace it
            }
        }
    }
    return false;
}

private static void reducedDomain(int[][] sudokuBoard, int i, int j) {

}

private static boolean checkConstraints(int[][] sudokuBoard,
```

```
                                int row, int col,
                                int number)
{
    for (int d = 0; d < sudokuBoard.length; d++)
    {
        //if number is already present in that row, return false
        if (sudokuBoard[row][d] == number)
        {
            if (d != col) return false;
        }
    }

    for (int r = 0; r < 9; r++)
    {
        //if number is already present in that column, return false
        if (sudokuBoard[r][col] == number)
        {
            if (r != row) return false;
        }
    }

    int sqrt = (int) Math.sqrt(9);
    int unitRow = row - row % sqrt;
    int unitColumn = col - col % sqrt;

    //if number is already present in that 3x3 square, return false
    for (int r = unitRow;
         r < unitRow + sqrt; r++)
    {
        for (int d = unitColumn;
             d < unitColumn + sqrt; d++)
        {
            if (sudokuBoard[r][d] == number)
            {
                if (r != row && d != col) return false;
            }
        }
    }

    // if all the constraints are satisfied, return true
    return true;
}

private static void showSolution(int[][] sudokuBoard)
{
    //showing the solution output on the GUI
    for (int r = 0; r < 9; r++)
    {
        for (int d = 0; d < 9; d++)
        {
            //System.out.print(sudokuBoard[r][d]);
            buttonGrid[r][d].setText(String.valueOf(sudokuBoard[r][d]));
```

```
        }
      }
   }

   //implementing the selectedColors function
   private void selectedColors() {
      for (int i = 0; i <3 ; i++) {
         for (int j = 0; j <3 ; j++) {
            buttonGrid[i][j].setBackground(Color.lightGray);
         }
      }
      for (int i = 0; i <3 ; i++) {
         for (int j = 6; j <9 ; j++) {
            buttonGrid[i][j].setBackground(Color.lightGray);
         }
      }
      for (int i = 6; i <9 ; i++) {
         for (int j = 0; j <3 ; j++) {
            buttonGrid[i][j].setBackground(Color.lightGray);
         }
      }
      for (int i = 6; i <9 ; i++) {
         for (int j = 6; j <9 ; j++) {
            buttonGrid[i][j].setBackground(Color.lightGray);
         }
      }
      for (int i = 3; i <6 ; i++) {
         for (int j = 3; j <6 ; j++) {
            buttonGrid[i][j].setBackground(Color.lightGray);
         }
      }
   }


   private void assignBorders() {      //messy function of assigning the borders to selected buttons
      for (int i = 0; i< buttonGrid.length; i++){
         for (int j=0; j<81;j++){
            if ((i==0 && j==2)||(i==0 && j==5)||(i==1 && j==2)|| (i==1 && j==5)
               ||(i==7 && j==2) || (i==7 && j==5) || (i==4 && j==2) || (i==4 && j==5)
               ||(i==8 && j==2) || (i==8 && j==5))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(1,1,1,2,Color.BLACK));
            else if ((i==0 && j==3)||(i==0 && j==6)||(i==1 && j==3)|| (i==1 && j==6)
                  ||(i==7 && j==3) || (i==7 && j==6) || (i==4 && j==3) || (i==4 && j==6)
                  ||(i==8 && j==3) || (i==8 && j==6))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(1,2,1,1,Color.BLACK));
            else if ((i==3 && j==0)||(i==3 && j==1)||(i==3 && j==4)|| (i==3 && j==7)
                  ||(i==3 && j==8) || (i==6 && j==0)||(i==6 && j==1)||(i==6 && j==4)|| (i==6 &&
j==7)
                  ||(i==6 && j==8))
```

```
buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(2,1,1,1,Color.BLACK));
            else if ((i==2 && j==0) ||(i==2 && j==1) ||(i==2 && j==4) ||(i==2 && j==7) ||
                  (i==2 && j==8) ||(i==5 && j==0) ||(i==5 && j==1) ||(i==5 && j==4) ||
                  (i==5 && j==7) ||(i==5 && j==8))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(1,1,2,1,Color.BLACK));
            else if ((i==2 && j==2) ||(i==2 && j==5) ||(i==5 && j==2) ||(i==5 && j==5))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(1,1,2,2,Color.BLACK));
            else if ((i==2 && j==3) ||(i==2 && j==6) ||(i==5 && j==3) ||(i==5 && j==6))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(1,2,2,1,Color.BLACK));
            else if ((i==3 && j==2) ||(i==3 && j==5) ||(i==6 && j==2) ||(i==6 && j==5))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(2,1,1,2,Color.BLACK));
            else if ((i==3 && j==3) ||(i==3 && j==6) ||(i==6 && j==3) ||(i==6 && j==6))

buttonGrid[i][j].setBorder(BorderFactory.createMatteBorder(2,2,1,1,Color.BLACK));
        }
    }
  }
}
```