

**PROBLEM 1**:

import java.util.Scanner;

public class Question1 {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

System.out.println("Enter the Adhaar ID");

int AdhaarID=sc.nextInt();

System.out.println("Enter the country name,district name,state name,person name");

String country=sc.nextLine();

String district=sc.nextLine();

String state=sc.nextLine();

String person=sc.nextLine();

Country c=new Country();

State s=new State();

Person p=new Person();

District d=new District();

c.setcName(country);

s.setsName(state);

d.setdName(district);

p.setpName(person);

p.setpAdhaarID(AdhaarID);

System.out.println("Country Name:" +c.getcName());

c.getCountryDetails();

System.out.println("State Name:" +s.getsName());

s.getStateDetails();

System.out.println("District Name:" +d.getdName());

d.getDistrictDetails();

System.out.println("Person Name:" +p.getpName());

System.out.println("Person ID:" +p.getpAdhaarID());

p.getPersonDetails();

MP m=new MP();

m.getMPDetails();

MLA mla=new MLA();

mla.getMLADetails();

DM dm=new DM();

dm.getDMDetails();

PM pm=new PM();

pm.getPMDetails();

CM cm=new CM();

cm.getCMDetails();

}

}

class Country{

State st;

PM pm;

private String cName;

Country(){}

Country(String cName){

this.cName=cName;

}

protected String getcName() {

return cName;

}

protected void setcName(String cName) {

this.cName = cName;

}

public void getCountryDetails(){

System.out.println("Method to get Country Details");

}

}

class State{

District dis;

CM cma;

private String sName;

State(){}

State(String sName){

this.sName=sName;

}

protected String getsName() {

return sName;

}

protected void setsName(String sName) {

this.sName = sName;

}

public void getStateDetails(){

System.out.println("Method to get State Details");

}

}

class District{

DM dma;

private String dName;

District(){}

District(String dName){

this.dName=dName;

}

protected String getdName() {

return dName;

}

protected void setdName(String dName) {

this.dName = dName;

}

public void getDistrictDetails(){

System.out.println("Method to get District Details");

}

}

class Person{

private String pName;

private int pAdhaarID;

Person(){}

Person(String pName,int pAdhaarID){

this.pName=pName;

this.pAdhaarID=pAdhaarID;

}

protected String getpName() {

return pName;

}

protected void setpName(String pName) {

this.pName = pName;

}

protected int getpAdhaarID() {

return pAdhaarID;

}

protected void setpAdhaarID(int pAdhaarID) {

this.pAdhaarID = pAdhaarID;

}

public void getPersonDetails(){

System.out.println("Method to get Person Details");

}

}

class MP extends Person{

public void getMPDetails(){

System.out.println("Method to get MP Details");

}

}

class MLA extends Person{

public void getMLADetails(){

System.out.println("Method to get MLA Details");

}

}

class DM extends Person{

public void getDMDetails(){

System.out.println("Method to get DM Details");

}

}

class PM extends MP{

public void getPMDetails(){

System.out.println("Method to get PM Details");

}

}

class CM extends MLA{

public void getCMDetails(){

System.out.println("Method to get CM Details");

}

}

**PROBLEM 2:**

package model;

import java.util.\*;

public class Sudoku implements SudokuConstants {

// generate a new solution

public static int[][] generateGame() {

int[][] game = new int[BOARD\_SIZE][BOARD\_SIZE];

// randomly generate the first row

game[0] = genRandomRow();

if ( !canBeFilled( 0, game ) )

return null;

return game;

}

// uses backtracking to come up with a solution

public static boolean canBeFilled( int index, int[][] cells ) {

if ( index > (BOARD\_SIZE \* BOARD\_SIZE - 1) )

return true;

int col = index % BOARD\_SIZE;

int row = index / BOARD\_SIZE;

if ( cells[row][col] != 0 )

return canBeFilled( index+1, cells);

int[] randomRow = genRandomRow();

for ( int i=0; i<BOARD\_SIZE; i++ ) {

if ( isValid( row, col, randomRow[i], cells ) ) {

cells[row][col] = randomRow[i];

if ( canBeFilled( index+1, cells) )

return true;

}

}

// no possible solution -> backtrack

cells[row][col] = 0;

return false;

}

// checks whether a value place at cells[row][col] is valid

public static boolean isValid( int row, int col, int value, int[][] cells ) {

// check whether there is a duplicate value in the row

for ( int i=0; i<BOARD\_SIZE; i++ ) {

if ( i == col )

continue;

if ( value == cells[row][i] ) {

return false;

}

}

// check whether there is a duplicate value in the column

for ( int i=0; i<BOARD\_SIZE; i++ ) {

if ( i == row )

continue;

if ( value == cells[i][col]) {

return false;

}

}

int sectRow = (row / BLOCK\_SIZE) \* BLOCK\_SIZE;

int sectCol = (col / BLOCK\_SIZE) \* BLOCK\_SIZE;

// check whether there is a duplicate value in the block

for ( int i=0; i<BLOCK\_SIZE; i++ ) {

for ( int j=0; j<BLOCK\_SIZE; j++ ) {

int rowOffset = sectRow + i;

int colOffset = sectCol + j;

if ( rowOffset == row && colOffset == col )

continue;

if ( value == cells[rowOffset][colOffset]) {

return false;

}

}

}

return true;

}

// generates an array with the integers 1 to 9 randomly placed

public static int[] genRandomRow() {

int[] row = new int[BOARD\_SIZE];

for ( int i=0; i<row.length; i++ ) {

row[ i ] = i+1;

}

Random rand = new Random();

for ( int i=row.length-1; i>0; i-- ) {

int j = rand.nextInt(i+1);

if ( i == j )

continue;

int temp = row[i];

row[i] = row[j];

row[j] = temp;

}

return row;

}

}

package model;

import java.util.\*;

import controller.\*;

public class GameStatus extends Observable implements SudokuConstants {

public static final int UNMODIFIABLE = -1;

public static final int HAS\_NO\_CONFLICT = 0;

public static final int HAS\_CONFLICT = 1;

private int[][] solution;

private int[][] game;

private int[][] boardStatus;

private CellLocation currCell;

private CellLocation prevCell;

private boolean isConnected;

public GameStatus() {

solution = new int[ BOARD\_SIZE ][BOARD\_SIZE];

game = new int[BOARD\_SIZE][BOARD\_SIZE];

boardStatus = new int[BOARD\_SIZE][BOARD\_SIZE];

currCell = new CellLocation( CellLocation.INVALID, CellLocation.INVALID );

prevCell = new CellLocation( CellLocation.INVALID, CellLocation.INVALID );

isConnected = false;

}

public void startNewGame( int level ) {

do {

solution = Sudoku.generateGame();

} while ( solution == null );

game = new int[BOARD\_SIZE][BOARD\_SIZE];

boardStatus = new int[BOARD\_SIZE][BOARD\_SIZE];

currCell.clear();

prevCell.clear();

setDifficulty( level );

NewGameState state = new NewGameState( game );

setChanged();

notifyObservers( state );

}

public void startReceivedGame( int[][] newGame, int[][] newBoardStatus ) {

game = newGame;

boardStatus = newBoardStatus;

currCell.clear();

prevCell.clear();

ReceivedGameState state = new ReceivedGameState( game, boardStatus );

setChanged();

notifyObservers( state );

}

public int[][] getGame() {

return game;

}

private void setDifficulty( int level ) {

int[] randomRow;

for ( int sect=0; sect<BOARD\_SIZE; sect++ ) {

randomRow = Sudoku.genRandomRow();

int startRow = (sect / BLOCK\_SIZE) \* BLOCK\_SIZE;

int startCol = (sect % BLOCK\_SIZE) \* BLOCK\_SIZE;

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

int offset = randomRow[i] - 1;

int rowOffset = offset / BLOCK\_SIZE;

int colOffset = offset % BLOCK\_SIZE;

game[startRow+rowOffset][startCol+colOffset] = solution[startRow+rowOffset][startCol+colOffset];

boardStatus[startRow+rowOffset][startCol+colOffset] = UNMODIFIABLE;

}

}

}

public void setFocus( int row, int col ) {

if ( boardStatus[row][col] == UNMODIFIABLE ) {

return;

}

if ( currCell.isEqualTo( new CellLocation( row, col ) ) )

return;

prevCell.copyFrom( currCell );

currCell.setLoc( row, col );

CellFocusedState state = new CellFocusedState( currCell, prevCell );

setChanged();

notifyObservers( state );

}

public CellLocation getCurrentCell() {

return currCell;

}

public CellLocation getPreviousCell() {

return prevCell;

}

public void clearNumber( int row, int col ) {

if ( boardStatus[row][col] == UNMODIFIABLE )

return;

setFocus( row, col );

game[row][col] = 0;

ValueClearedState state = new ValueClearedState( new CellLocation( row, col) );

setChanged();

notifyObservers( state );

checkGame();

}

public void setNumberInGame( CellLocation loc, int nbr ) {

setNumberInGame( loc.getRow(), loc.getCol(), nbr );

}

public void setNumberInGame( int row, int col, int nbr ){

game[row][col] = nbr;

ValueChangedState state = new ValueChangedState( new CellLocation( row, col), nbr );

setChanged();

notifyObservers( state );

checkGame();

}

public void updateReceivedNbr( int row, int col, int nbr ){

game[row][col] = nbr;

ValueReceivedState state = new ValueReceivedState( new CellLocation( row, col), nbr );

setChanged();

notifyObservers( state );

checkGame();

}

public int getNumberInGame( int row, int col ){

return game[row][col];

}

private synchronized void checkGame() {

boolean isSolved = true;

for ( int row=0; row<BOARD\_SIZE; row++ ) {

for (int col=0; col<BOARD\_SIZE; col++ ) {

int value = game[row][col];

if ( value == 0 ) {

isSolved = false;

boardStatus[row][col] = HAS\_NO\_CONFLICT;

continue;

}

int status = boardStatus[row][col];

if ( status == UNMODIFIABLE ) {

continue;

}

if ( !Sudoku.isValid( row, col, value, game )) {

isSolved = false;

boardStatus[row][col] = HAS\_CONFLICT;

} else {

boardStatus[row][col] = HAS\_NO\_CONFLICT;

}

}

}

if ( isSolved ) {

lockBoard();

GameSolvedState solved = new GameSolvedState();

setChanged();

notifyObservers( solved );

return;

}

ConflictingValuesState conflict = new ConflictingValuesState( boardStatus );

setChanged();

notifyObservers( conflict );

}

private void lockBoard() {

// set all cells to unmodifiable

for ( int row=0; row<BOARD\_SIZE; row++ ) {

for ( int col=0; col<BOARD\_SIZE; col++ ) {

boardStatus[row][col] = UNMODIFIABLE;

}

}

}

public void hasConnection() {

isConnected = true;

ConnectedState state = new ConnectedState();

setChanged();

notifyObservers( state );

}

public void hasNoConnection() {

isConnected = false;

LostConnectionState state = new LostConnectionState();

setChanged();

notifyObservers( state );

}

public String serialize() {

StringBuffer buffer = new StringBuffer();

for ( int row=0; row<BOARD\_SIZE; row++ ) {

for ( int col=0; col<BOARD\_SIZE; col++ ) {

buffer.append( row + "," + col + "," + game[row][col] + "," + boardStatus[row][col] + ";" );

}

}

return buffer.toString();

}

public void deserialize( String data ) {

int[][] newGame = new int[BOARD\_SIZE][BOARD\_SIZE];

int[][] newBoardStatus = new int[BOARD\_SIZE][BOARD\_SIZE];

String[] tokens = data.split( ";" );

for ( String token : tokens ) {

String[] fields = token.split(",");

int row = Integer.parseInt( fields[0] );

int col = Integer.parseInt( fields[1] );

int nbr = Integer.parseInt( fields[2] );

int status = Integer.parseInt( fields[3] );

newGame[row][col] = nbr;

newBoardStatus[row][col] = status;

}

startReceivedGame( newGame, newBoardStatus );

}

}

package model;

import java.awt.Color;

/\*\*

\* This file declares several constants that are useful in

\* the program in different modules. Any class that

\* implements this interface can use these constants.

\*/

public interface SudokuConstants {

/\*\*

\* The size of the game board, which is also the

\* number of cells in a block. The board should

\* always be square, with equal height and width.

\*/

public static final int BOARD\_SIZE = 9;

/\*\*

\* The height (how many rows) of a single block.

\* The width of a block should be BOARD\_SIZE / BLOCK\_HEIGHT

\*/

public static final int BLOCK\_HEIGHT = 3;

/\*\*

\* The default port number for both server and client.

\* It is shown as the default value for the text fields.

\*/

public static final int DEFAULT\_PORT = 31416;

/\*\*

\* The default IP address to connect for client.

\* It is shown as the default value for the text fields.

\*/

public static final String DEFAULT\_IP = "127.0.0.1";

/\*\*

\* The default difficulty level. It is shown as the default

\* value for the difficulty text field. You can change this

\* value to fit your own difficulty level system.

\*/

public static final int DEFAULT\_DIFFICULTY = 5;

public static final int BLOCK\_SIZE = BOARD\_SIZE / BLOCK\_HEIGHT;

public static final String LOGIN\_WINDOW\_NAME = "Login";

public static final int LOGIN\_WINDOW\_HEIGHT = 150;

public static final int LOGIN\_WINDOW\_WIDTH = 480;

public static final String LAUNCH\_BTN\_NAME = "Launch!";

public static final String JOIN\_BTN\_NAME = "Join!";

public static final String GAME\_WINDOW\_NAME = "Sudoku";

public static final int HOST\_WINDOW\_HEIGHT = 500;

public static final int HOST\_WINDOW\_WIDTH = 500;

public static final int GUEST\_WINDOW\_HEIGHT = 460;

public static final int GUEST\_WINDOW\_WIDTH = 500;

public static final String NEW\_GAME\_BTN\_NAME = "New Game";

public static final int MIN\_DIFFICULTY = 1;

public static final int MAX\_DIFFICULTY = 8;

public static final Color USER\_COLOR = Color.BLACK;

public static final Color FOREIGN\_COLOR = Color.YELLOW;

public static final Color BLOCK\_COLOR1 = new Color( 0, 204, 255 );

public static final Color BLOCK\_COLOR2 = new Color( 0, 255, 204 );

public static final Color BLOCK\_BORDER\_COLOR = Color.DARK\_GRAY;

public static final Color CELL\_BORDER\_COLOR = Color.GRAY;

public static final Color UNMODIFIABLE\_COLOR = Color.GRAY;

public static final Color FOCUSED\_COLOR = Color.YELLOW;

public static final Color CONFLICT\_COLOR = Color.RED;

public static final int BOARD\_HEIGHT = 400;

public static final int BOARD\_WIDTH = 400;

public static final int CELL\_HEIGHT = 40;

public static final int CELL\_WIDTH = 40;

public static final int CELL\_NUMBER\_SIZE = 20;

package model;

public class CellLocation {

public static final int INVALID = -1;

private int row;

private int col;

public CellLocation( int row, int col ) {

this.row = row;

this.col = col;

}

public int getRow() {

return row;

}

public void setRow( int row ) {

this.row = row;

}

public int getCol() {

return col;

}

public void setCol( int col ) {

this.col = col;

}

public void setLoc( int row, int col ) {

this.row = row;

this.col = col;

}

public void clear() {

row = INVALID;

col = INVALID;

}

public boolean isValid() {

if ( row == INVALID || col == INVALID )

return false;

return true;

}

public boolean isEqualTo( CellLocation loc ) {

if ( row != loc.getRow() )

return false;

if ( col != loc.getCol() )

return false;

return true;

}

public void copyFrom( CellLocation loc ) {

row = loc.getRow();

col = loc.getCol();

}

}

public static final String NEW\_GAME\_MSG = "new\_game";

public static final String NEW\_NBR\_MSG = "new\_nbr";

public static final String ACK\_MSG = "OK";

}

import java.io.\*;

import java.net.\*;

import java.util.Observable;

import java.util.Observer;

import model.\*;

public class ClientThread extends Thread implements SudokuConstants, Observer {

Socket socket = null;

PrintWriter out;

BufferedReader in;

GameStatus gameStatus;

public ClientThread( Socket socket, GameStatus gameStatus ) {

this.socket = socket;

this.gameStatus = gameStatus;

}

public void run() {

try {

this.gameStatus.hasConnection();

this.gameStatus.addObserver( this );

out = new PrintWriter( socket.getOutputStream(), true );

in = new BufferedReader( new InputStreamReader( socket.getInputStream() ));

// upon connection, receive data for current game

String fromServer = in.readLine();

if ( fromServer.equals( NEW\_GAME\_MSG ) ) {

fromServer = in.readLine();

parseNewGameData( fromServer );

out.println( ACK\_MSG );

out.flush();

}

while ( (fromServer = in.readLine()) != null ) {

if ( fromServer.equals( NEW\_GAME\_MSG ) ) {

fromServer = in.readLine();

parseNewGameData( fromServer );

} else if (fromServer.equals( NEW\_NBR\_MSG ) ) {

fromServer = in.readLine();

setNumber( fromServer );

}

}

out.close();

in.close();

socket.close();

this.gameStatus.deleteObserver( this );

this.gameStatus.hasNoConnection();

} catch( IOException e ) {

System.out.println( "Error:" + e.getMessage() );

}

}

private void parseNewGameData( String data ) {

gameStatus.deserialize( data );

}

private void setNumber( String data ) {

String[] fields = data.split(",");

int row = Integer.parseInt( fields[0] );

int col = Integer.parseInt( fields[1] );

int nbr = Integer.parseInt( fields[2] );

gameStatus.updateReceivedNbr( row, col, nbr);

}

private void sendNumber( CellLocation loc, int nbr ) {

if ( out == null )

return;

out.println( NEW\_NBR\_MSG );

out.println( loc.getRow() + "," + loc.getCol() + "," + nbr );

out.flush();

}

public void update( Observable o, Object arg ) {

if ( arg instanceof ValueChangedState ) {

ValueChangedState state = (ValueChangedState) arg;

sendNumber( state.loc, state.newValue );

return;

}

if ( arg instanceof ValueClearedState ) {

ValueClearedState state = (ValueClearedState) arg;

sendNumber( state.loc, 0 );

return;

}

}

}

import java.io.\*;

import java.net.\*;

import java.util.Observable;

import java.util.Observer;

import model.\*;

public class ServerThread extends Thread implements SudokuConstants, Observer {

ServerSocket server = null;

PrintWriter out;

BufferedReader in;

GameStatus gameStatus;

public ServerThread( ServerSocket server, GameStatus gameStatus ) {

this.server = server;

this.gameStatus = gameStatus;

}

public void run() {

try {

Socket socket = server.accept();

this.gameStatus.hasConnection();

this.gameStatus.addObserver( this );

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

out = new PrintWriter(socket.getOutputStream(), true);

// upon connection, send over current game

sendEntireGame();

String fromClient = in.readLine();

while ( (fromClient = in.readLine()) != null ) {

if (fromClient.equals( NEW\_NBR\_MSG ) ) {

fromClient = in.readLine();

setNumber( fromClient );

}

}

out.close();

in.close();

socket.close();

this.gameStatus.deleteObserver( this );

this.gameStatus.hasNoConnection();

} catch( Exception e ) {

System.out.println( "Error: " + e.getMessage() );

}

}

private void sendEntireGame() {

if ( out == null )

return;

out.println( NEW\_GAME\_MSG );

out.println( gameStatus.serialize() );

out.flush();

}

private void sendNumber( CellLocation loc, int nbr ) {

if ( out == null )

return;

out.println( NEW\_NBR\_MSG );

out.println( loc.getRow() + "," + loc.getCol() + "," + nbr );

out.flush();

}

private void setNumber( String data ) {

String[] fields = data.split(",");

int row = Integer.parseInt( fields[0] );

int col = Integer.parseInt( fields[1] );

int nbr = Integer.parseInt( fields[2] );

gameStatus.updateReceivedNbr( row, col, nbr);

}

public void update( Observable o, Object arg ) {

if ( arg instanceof NewGameState ) {

NewGameState state = (NewGameState) arg;

sendEntireGame();

return;

}

if ( arg instanceof ValueChangedState ) {

ValueChangedState state = (ValueChangedState) arg;

sendNumber( state.loc, state.newValue );

return;

}

if ( arg instanceof ValueClearedState ) {

ValueClearedState state = (ValueClearedState) arg;

sendNumber( state.loc, 0 );

return;

}

}

}