**Endterm Project Report submission**

**Mordvintsev Nikita**

**Astana IT University**

**PhD. Aitmukhanbetova Elvira**

**18 February 2022**

**Outline:**

This document contains: the proofs for Endterm project’s criteria following, screenshots how it is working, and schema of project.

// I don't have a lot of comments in my code. In requirements written, paste the class code as text.

* Title
* Outline
* Idea (Introduction)
* Following criteria
* Structure
* Classes
* Screenshots
* Conclusion

PrSc – Project screen (Screen from project application)

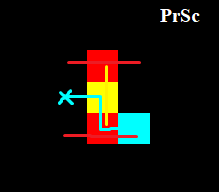
**Idea:**

Score – gaming convention, quantitative privilege.

Grid – two-dimensional array of squares. Every square have a type which affects on addition to score.

Square behaviors:

|  |  |
| --- | --- |
| Square | Behavior |
| Red | Increases score more if have a chain of squares to right and left. |
| Yellow | Increases score more if have a chain of squares to up and down. |
| Cyan | Increases score more if have a chain of squares to diagonals. |



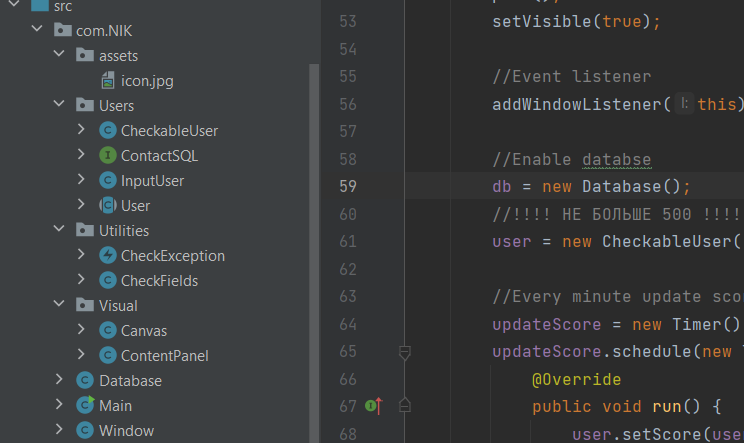
Client application that sends every time gap user’s grid and score data.

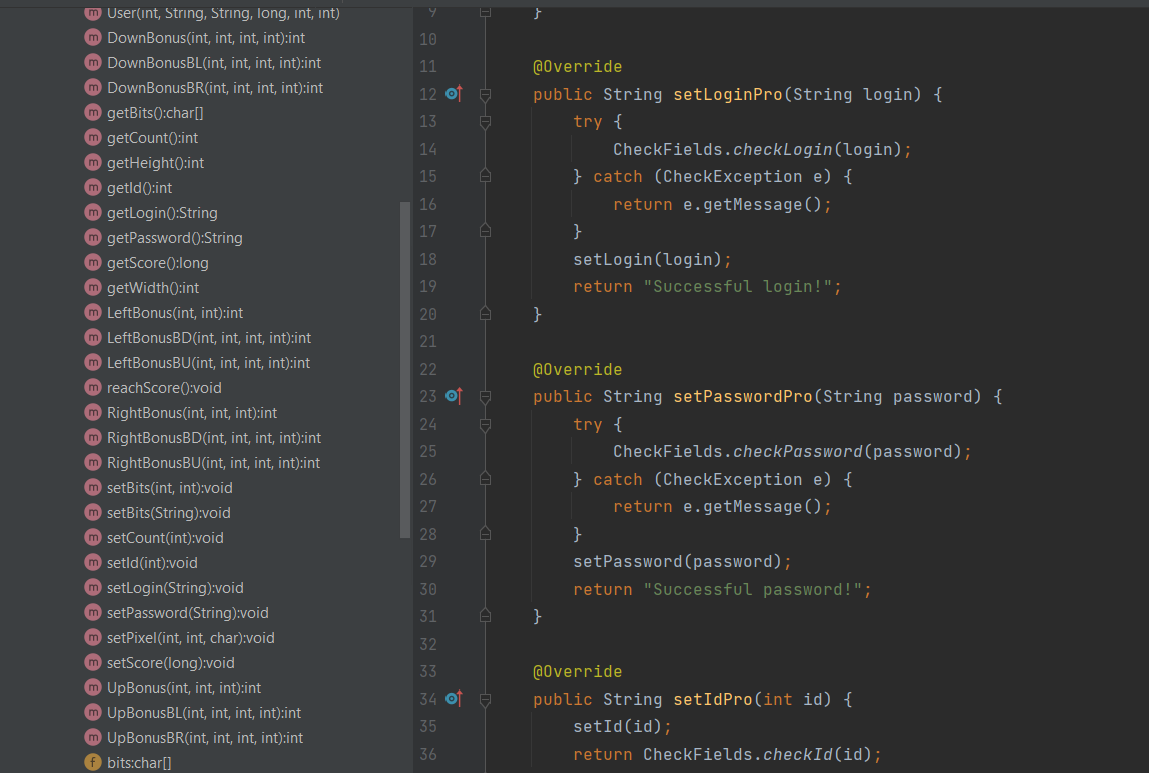
Data that active sends to SQL database every time gap and when window closes: Text (Grid of characters), score, count of squares that could be placed.

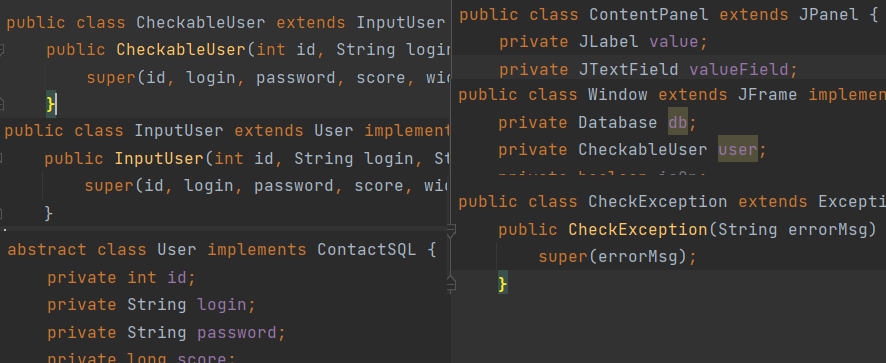
**Following criteria:**

“Use the capabilities of OOP”:

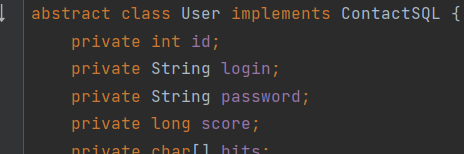
* Class and objects:



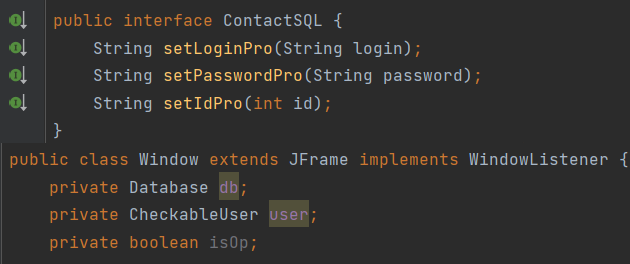
* Encapsulation
* Inheritance:



* Polymorphism:
  + Abstract class:



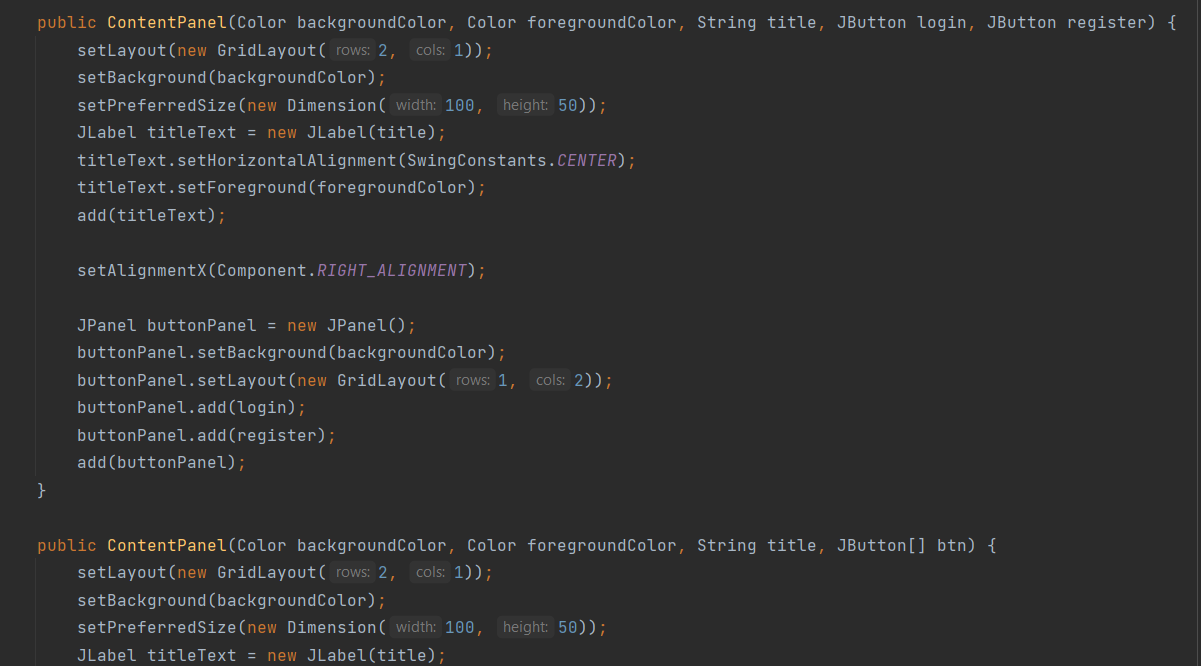
* + Interface:



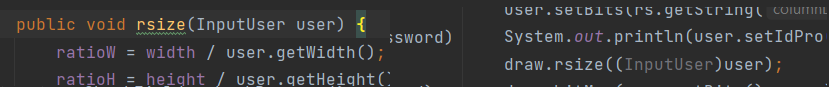
* + Overriding:



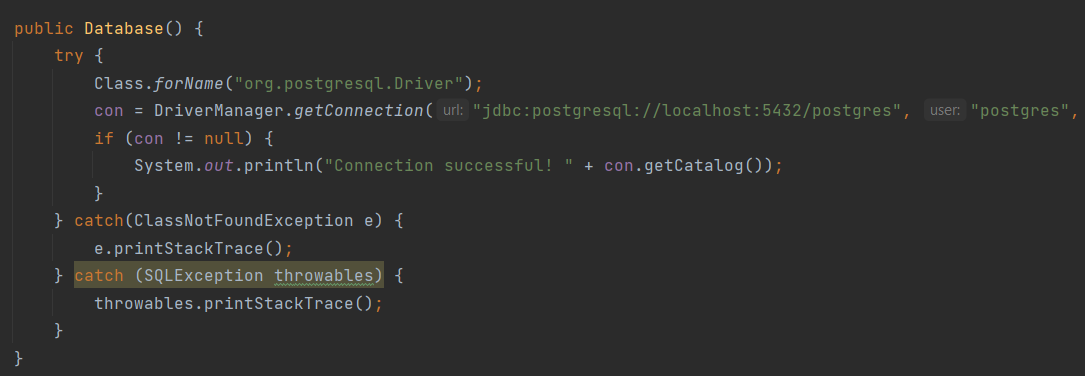
* + Overloading:

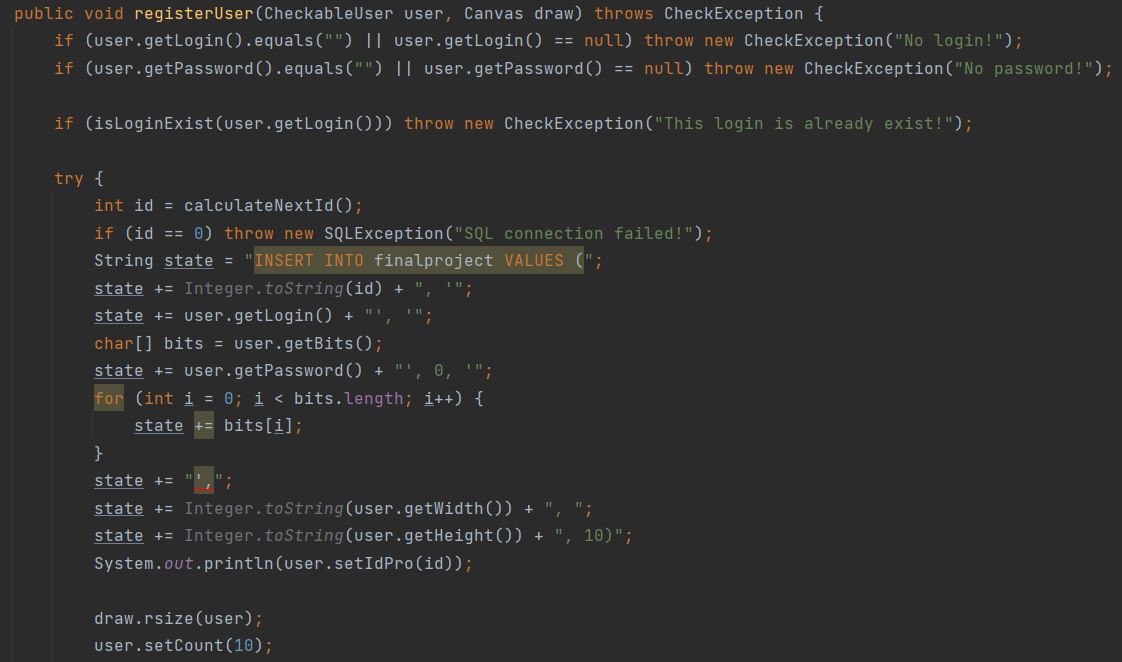


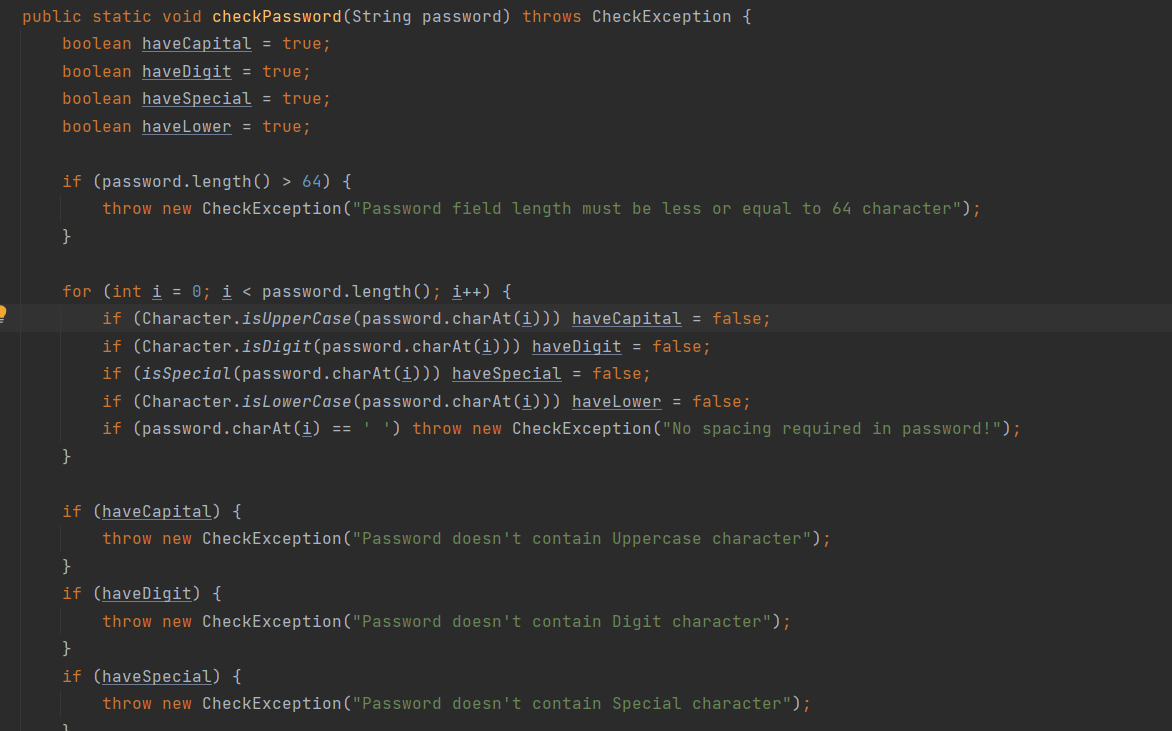
* + Parent reference to Child:

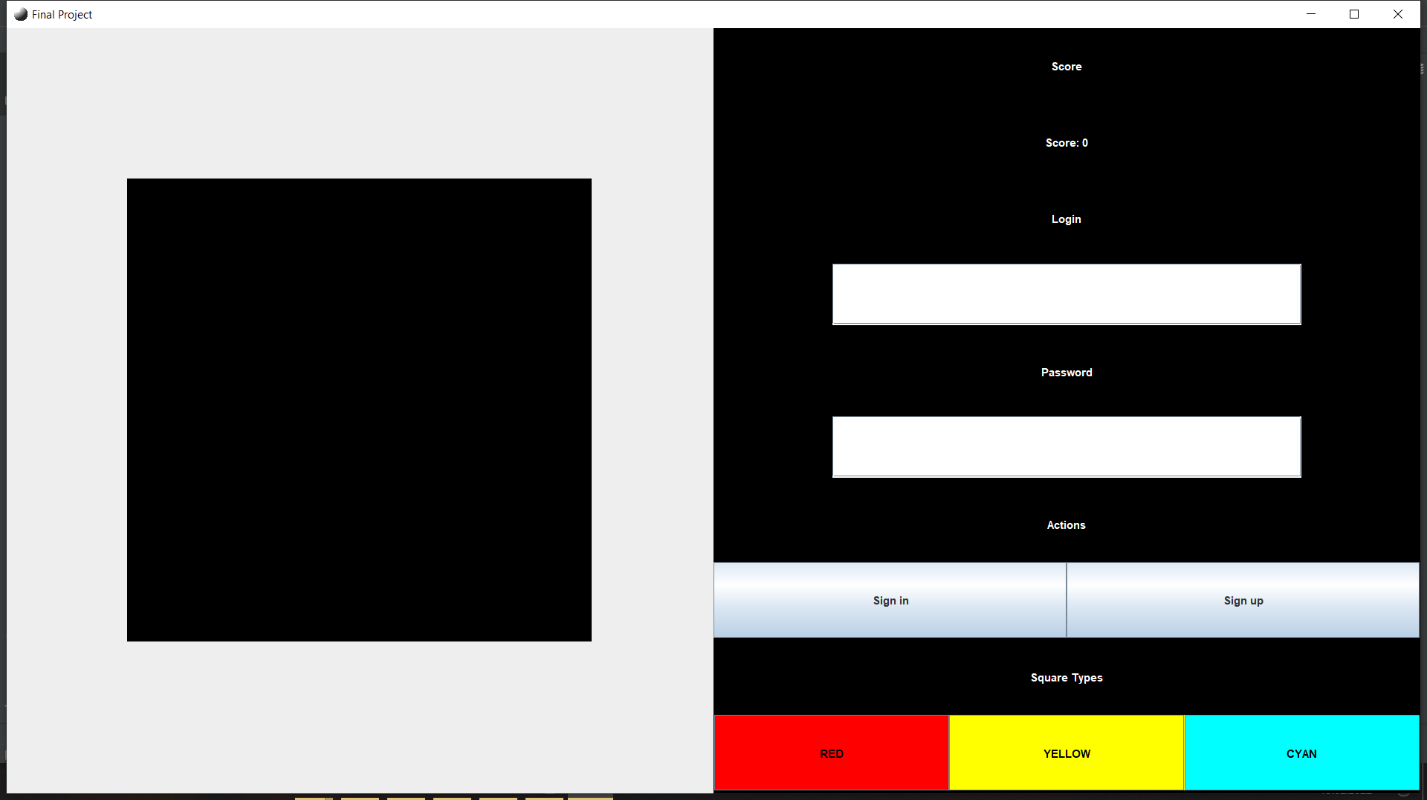


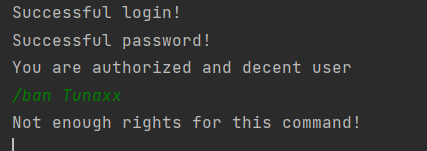
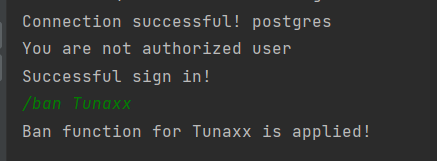
* + JDBC:



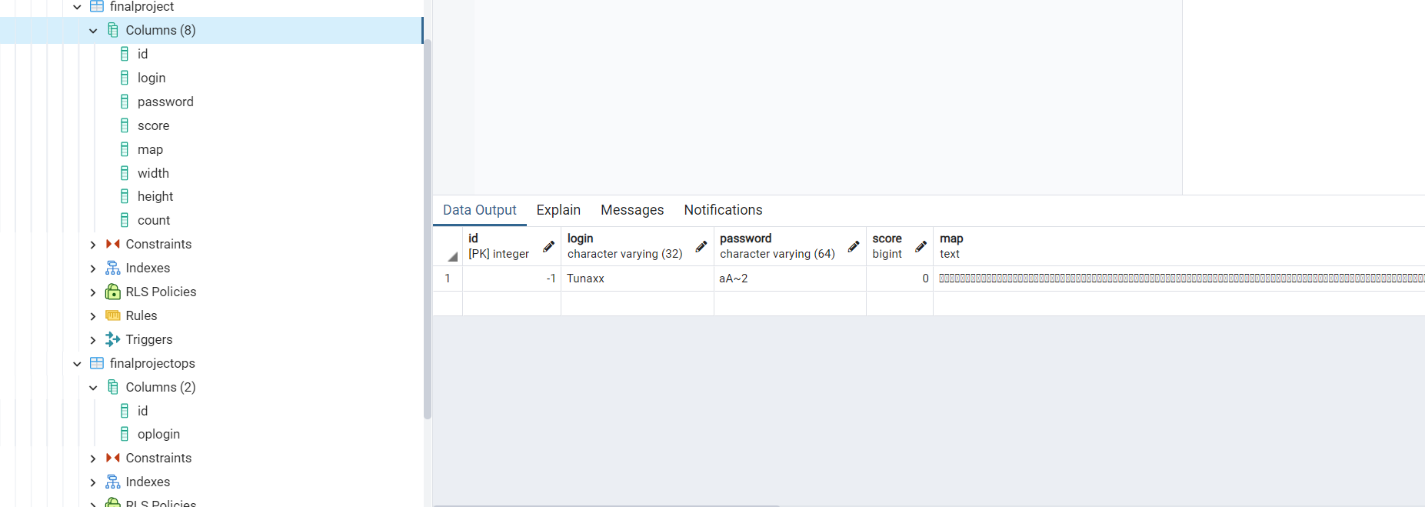


* + Exception handling:
  + GUI:





* + DBMS: SQL postgres



**Structure:**

ContactSQL(Interface) setLogin, Password, Id with handling exceptions and output messages

(abstract) User -> Input user (With interface) -> Checkable user(Usable class)

CheckException(extends Exception)

CheckFields(class which checks password and login field and send comments)

Canvas(extends JPanel) drawable part, grid

ContentPanel(extends JPanel) UI with JLabels, JButtons

Main: Runnable(Window(extends JFrame)) and Timer(For accept commands from console)

Window creates object from:

Database(Class with database manipulations)

CheckableUser(Active user)

SQL:

Finalproject(table):

|  |  |
| --- | --- |
| Id | Integer |
| Login | Varchar(32) |
| Password | Varchar(64) |
| Score | Bigint |
| Map | Text |
| Width | Integer |
| Height | Integer |
| Count | Integer |

Finalprojectops(table):

|  |  |
| --- | --- |
| Id | Integer |
| oplogin | Varchar(32) |

**Classes:**

**User**

package com.NIK.Users;  
  
abstract class User implements ContactSQL {  
 private int id;  
 private String login;  
 private String password;  
 private long score;  
 private char[] bits;  
 private int width;  
 private int height;  
 private int count;  
  
 public User(int id, String login, String password, long score, int width, int height) {  
 this.id = id;  
 this.login = login;  
 this.password = password;  
 this.score = score;  
 this.bits = new char[width \* height];  
 this.width = width;  
 this.height = height;  
 for (int i = 0; i < this.bits.length; i++) {  
 bits[i] = 1;  
 }  
 }  
  
 public int getCount() {  
 return count;  
 }  
  
 public void setCount(int count) {  
 this.count = count;  
 }  
  
 public long getScore() {  
 return score;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
  
 public void setBits(int width, int height) {  
 this.width = width;  
 this.height = height;  
 this.bits = new char[width \* height];  
 for (int i = 0; i < this.bits.length; i++) {  
 bits[i] = 1;  
 }  
 }  
  
 public int getHeight() {  
 return height;  
 }  
  
 public void reachScore() {  
 int scorePP = 0;  
 for (int i = 0; i < height; i++) {  
 for (int j = 0; j < width; j++) {  
 int index = i \* width + j;  
 switch (bits[index]) {  
 case 'A':  
 {  
 //LEFT and RIGHT wire  
 scorePP += LeftBonus(index, j);  
 scorePP += RightBonus(index, j, width - 1);  
 break;  
 }  
 case 'B':  
 {  
 // UP and DOWN wire  
 scorePP += UpBonus(index, i, width);  
 scorePP += DownBonus(index, i, width, height - 1);  
 break;  
 }  
 case 'C':  
 {  
 // ^  
 // |<-^  
 // |<-\*  
 scorePP += LeftBonusBU(index, j, i, width);  
 scorePP += LeftBonusBD(index, j, i, width);  
 scorePP += RightBonusBU(index, j, i, width);  
 scorePP += RightBonusBD(index, j, i, width);  
 break;  
 }  
 }  
 }  
 }  
 setScore(getScore() + scorePP);  
 }  
  
 public int LeftBonus(int index, int j) {  
 if (j != 0) {  
 if (bits[index - 1] != '\u0001') return LeftBonus(index - 1, j - 1) + 1;  
 }  
 return 1;  
 }  
 public int RightBonus(int index, int j, int max) {  
 if (j != max) {  
 if (bits[index + 1] != '\u0001') return RightBonus(index + 1, j + 1, max) + 1;  
 }  
 return 1;  
 }  
  
 public int UpBonus(int index, int i, int decrement) {  
 if (i != 0) {  
 if (bits[index - decrement] != '\u0001') return UpBonus(index - decrement, i - 1, decrement) + 1;  
 }  
 return 1;  
 }  
 public int DownBonus(int index, int i, int decrement, int max) {  
 if (i != max) {  
 if (bits[index + decrement] != '\u0001') return DownBonus(index + decrement, i + 1, decrement, max) + 1;  
 }  
 return 1;  
 }  
  
 public int LeftBonusBD(int index, int j, int i, int decrement) {  
 if (j != 0) {  
 if (bits[index - 1] != '\u0001') return DownBonusBL(index - 1, j - 1, i, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int RightBonusBD(int index, int j, int i, int decrement) {  
 if (j != width - 1) {  
 if (bits[index + 1] != '\u0001') return DownBonusBR(index + 1, j + 1, i, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int LeftBonusBU(int index, int j, int i, int decrement) {  
 if (j != 0) {  
 if (bits[index - 1] != '\u0001') return UpBonusBL(index - 1, j - 1, i, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int RightBonusBU(int index, int j, int i, int decrement) {  
 if (j != width - 1) {  
 if (bits[index + 1] != '\u0001') return UpBonusBR(index + 1, j + 1, i, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int UpBonusBR(int index, int j, int i, int decrement) {  
 if (i != 0) {  
 if (bits[index - decrement] != '\u0001') return RightBonusBU(index - decrement, j, i - 1, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int UpBonusBL(int index, int j, int i, int decrement) {  
 if (i != 0) {  
 if (bits[index - decrement] != '\u0001') return LeftBonusBU(index - decrement, j, i - 1, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int DownBonusBL(int index, int j, int i, int decrement) {  
 if (i != height - 1) {  
 if (bits[index + decrement] != '\u0001') return LeftBonusBD(index + decrement, j, i + 1, decrement) \* 2;  
 }  
 return 1;  
 }  
 public int DownBonusBR(int index, int j, int i, int decrement) {  
 if (i != height - 1) {  
 if (bits[index + decrement] != '\u0001') return RightBonusBD(index + decrement, j, i + 1, decrement) \* 2;  
 }  
 return 1;  
 }  
  
 public void setPixel(int x, int y, char a) {  
 bits[y \* width + x] = a;  
 }  
  
 public int getWidth() {  
 return width;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public char[] getBits() {  
 return bits;  
 }  
  
 public void setScore(long score) {  
 this.score = score;  
 }  
  
 public void setLogin(String login) {  
 this.login = login;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public void setBits(String text) {  
 for (int i = 0; i < text.length(); i++) {  
 this.bits[i] = text.charAt(i);  
 }  
 }  
}

**InputUser**

package com.NIK.Users;  
  
public class InputUser extends User implements ContactSQL {  
 public InputUser(int id, String login, String password, long score, int width, int height) {  
 super(id, login, password, score, width, height);  
 }  
  
 @Override  
 public String setLoginPro(String login) {  
 setLogin(login);  
 return "";  
 }  
  
 @Override  
 public String setPasswordPro(String password) {  
 setPassword(password);  
 return "";  
 }  
  
 @Override  
 public String setIdPro(int id) {  
 setId(id);  
 return "";  
 }  
}

**CheckableUser**

package com.NIK.Users;  
  
import com.NIK.Utilities.CheckException;  
import com.NIK.Utilities.CheckFields;  
  
public class CheckableUser extends InputUser {  
 public CheckableUser(int id, String login, String password, long score, int width, int height) {  
 super(id, login, password, score, width, height);  
 }  
  
 @Override  
 public String setLoginPro(String login) {  
 try {  
 CheckFields.*checkLogin*(login);  
 } catch (CheckException e) {  
 return e.getMessage();  
 }  
 setLogin(login);  
 return "Successful login!";  
 }  
  
 @Override  
 public String setPasswordPro(String password) {  
 try {  
 CheckFields.*checkPassword*(password);  
 } catch (CheckException e) {  
 return e.getMessage();  
 }  
 setPassword(password);  
 return "Successful password!";  
 }  
  
 @Override  
 public String setIdPro(int id) {  
 setId(id);  
 return CheckFields.*checkId*(id);  
 }  
}

**ContactSQL – Interface**

package com.NIK.Users;  
  
public interface ContactSQL {  
 String setLoginPro(String login);  
 String setPasswordPro(String password);  
 String setIdPro(int id);  
}

**CheckExceptions**

package com.NIK.Utilities;  
  
public class CheckException extends Exception {  
 public CheckException(String errorMsg) {  
 super(errorMsg);  
 }  
}

**CheckFields**

package com.NIK.Utilities;  
  
public class CheckFields {  
 public static void checkLogin(String login) throws CheckException {  
 if (login.length() > 32) {  
 throw new CheckException("Login field length must be less or equal to 32 character");  
 }  
 if (login.length() <= 0 || login.equals("")) {  
 throw new CheckException("There is no login!");  
 }  
 }  
  
 public static void checkPassword(String password) throws CheckException {  
 boolean haveCapital = true;  
 boolean haveDigit = true;  
 boolean haveSpecial = true;  
 boolean haveLower = true;  
  
 if (password.length() > 64) {  
 throw new CheckException("Password field length must be less or equal to 64 character");  
 }  
  
 for (int i = 0; i < password.length(); i++) {  
 if (Character.*isUpperCase*(password.charAt(i))) haveCapital = false;  
 if (Character.*isDigit*(password.charAt(i))) haveDigit = false;  
 if (*isSpecial*(password.charAt(i))) haveSpecial = false;  
 if (Character.*isLowerCase*(password.charAt(i))) haveLower = false;  
 if (password.charAt(i) == ' ') throw new CheckException("No spacing required in password!");  
 }  
  
 if (haveCapital) {  
 throw new CheckException("Password doesn't contain Uppercase character");  
 }  
 if (haveDigit) {  
 throw new CheckException("Password doesn't contain Digit character");  
 }  
 if (haveSpecial) {  
 throw new CheckException("Password doesn't contain Special character");  
 }  
 if (haveLower) {  
 throw new CheckException("Password doesn't contain Lowercase character");  
 }  
 }  
  
 public static boolean isSpecial(Character a) {  
 boolean isSpec = false;  
 String specials = "!\"#$%&'()\*+,`/:;<=>?@[\\]^~{|}\_-";  
 for (int i = 0; i < specials.length(); i++) {  
 if (Character.*compare*(a, specials.charAt(i)) == 0) isSpec = true;  
 }  
 return isSpec;  
 }  
  
 public static String checkId(int id) {  
 if (id < 0) return "You are in black list";  
 if (id == 0) return "You are not authorized user";  
 if (id > 0) return "You are authorized and decent user";  
 return "";  
 }  
}

**Canvas**

package com.NIK.Visual;  
  
import com.NIK.Users.CheckableUser;  
import com.NIK.Users.InputUser;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
  
public class Canvas extends JPanel {  
 private BufferedImage bi;  
 private ImageIcon canv;  
 private CheckableUser user;  
 private int ratioW;  
 private int ratioH;  
 private int width;  
 private int height;  
 private JLabel img;  
  
 private int dx = 0;  
 private int dy = 0;  
  
 public Canvas(int width, int height, CheckableUser user, int uwidth, int uheight) {  
 setLayout(new BorderLayout());  
  
 bi = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);  
 canv = new ImageIcon(bi);  
 img = new JLabel(canv);  
 img.setVerticalAlignment(SwingConstants.*CENTER*);  
 add(img);  
  
 ratioW = width / uwidth;  
 ratioH = height / uheight;  
 this.width = width;  
 this.height = height;  
  
 this.user = user;  
 }  
  
 public JLabel getImg() {  
 return img;  
 }  
  
 public void rsize(int w, int h) {  
 dx = ((w / 2) - width) / 2;  
 //30 - is Title panel of window  
 dy = (h - height - 30) / 2;  
 }  
  
 public void rsize(InputUser user) {  
 ratioW = width / user.getWidth();  
 ratioH = height / user.getHeight();  
 bi = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);  
 canv = new ImageIcon(bi);  
 img.setIcon(canv);  
 }  
  
 public void bitMap(char[] map, CheckableUser user) {  
 int w = user.getWidth();  
 for (int i = 0; i < user.getHeight(); i++) {  
 for (int j = 0; j < w; j++) {  
 switch (map[i \* w + j]) {  
 case 'A': {  
 rectangleDraw(j\*ratioH, i\*ratioW, ratioW, ratioH, Color.*RED*);  
 break;  
 }  
 case 'B': {  
 rectangleDraw(j\*ratioH, i\*ratioW, ratioW, ratioH, Color.*YELLOW*);  
 break;  
 }  
 case 'C': {  
 rectangleDraw(j\*ratioH, i\*ratioW, ratioW, ratioH, Color.*CYAN*);  
 break;  
 }  
 }  
 }  
 }  
 revalidate();  
 repaint();  
 }  
  
 public void setRect(int x, int y, char a, CheckableUser user) {  
 int x0 = x - x % ratioW - dx;  
 int y0 = y - y % ratioH - dy;  
  
 Color color = Color.*BLACK*;  
 switch (a) {  
 case 'A':  
 color = Color.*RED*;  
 break;  
 case 'B':  
 color = Color.*YELLOW*;  
 break;  
 case 'C':  
 color = Color.*CYAN*;  
 break;  
 case '\u0001':  
 color = Color.*BLACK*;  
 break;  
 }  
  
 rectangleDraw(x0, y0, ratioW, ratioH, color);  
 canv.setImage(bi);  
  
 user.setPixel(x0 / ratioW, y0 / ratioH, a);  
  
 revalidate();  
 repaint();  
 }  
  
 public boolean isRect(int x, int y, CheckableUser user) {  
 int x0 = x - x % ratioW - dx;  
 x0 /= ratioW;  
 int y0 = y - y % ratioH - dy;  
 y0 /= ratioH;  
 if (user.getBits()[y0 \* user.getWidth() + x0] != '\u0001') return true;  
 return false;  
 }  
  
 public void rectangleDraw(int x, int y, int width, int height, Color color) {  
 for (int i = 0; i < width; i++) {  
 for (int j = 0; j < height; j++) {  
 pixel(x + j, y + i, color);  
 }  
 }  
 }  
  
 public void setPixel(int x, int y, Color color) {  
 bi.setRGB(x, y, color.getRGB());  
 canv.setImage(bi);  
 }  
 public void pixel(int x, int y, Color color) {  
 bi.setRGB(x, y, color.getRGB());  
 }  
}

**ContentPanel**

package com.NIK.Visual;  
  
import javax.swing.\*;  
import javax.swing.border.Border;  
import javax.swing.border.EmptyBorder;  
import java.awt.\*;  
  
public class ContentPanel extends JPanel {  
 private JLabel value;  
 private JTextField valueField;  
 private JPasswordField valueFieldPassword;  
 public ContentPanel(Color backgroundColor, Color foregroundColor, String title, int type) {  
 setLayout(new GridLayout(2, 1));  
 setBackground(backgroundColor);  
 setPreferredSize(new Dimension(100, 50));  
 JLabel titleText = new JLabel(title);  
 titleText.setHorizontalAlignment(SwingConstants.*CENTER*);  
 titleText.setForeground(foregroundColor);  
 add(titleText);  
  
 setAlignmentX(Component.*RIGHT\_ALIGNMENT*);  
  
 switch (type) {  
 case 0: {  
 JPanel valueFieldPanel = new JPanel();  
 valueFieldPanel.setBackground(Color.*BLACK*);  
 valueFieldPanel.setLayout(new BoxLayout(valueFieldPanel, BoxLayout.*X\_AXIS*));  
 valueFieldPanel.setAlignmentX(Component.*RIGHT\_ALIGNMENT*);  
  
 setBorder(valueFieldPanel, 128, 8, 128, 8);  
  
 valueField = new JTextField();  
 valueField.setPreferredSize(new Dimension(50, 20));  
 valueField.setMinimumSize(new Dimension(10, 20));  
 valueField.setHorizontalAlignment(JTextField.*CENTER*);  
  
 valueFieldPanel.add(valueField);  
 add(valueFieldPanel);  
 break;  
 }  
 case 1: {  
 value = new JLabel("-----");  
 value.setForeground(foregroundColor);  
 value.setHorizontalAlignment(SwingConstants.*CENTER*);  
 add(value);  
 break;  
 }  
 case 2: {  
 break;  
 }  
 case 3: {  
 JPanel valueFieldPanel = new JPanel();  
 valueFieldPanel.setBackground(Color.*BLACK*);  
 valueFieldPanel.setLayout(new BoxLayout(valueFieldPanel, BoxLayout.*X\_AXIS*));  
 valueFieldPanel.setAlignmentX(Component.*RIGHT\_ALIGNMENT*);  
  
 setBorder(valueFieldPanel, 128, 8, 128, 8);  
  
 valueFieldPassword = new JPasswordField();  
 valueFieldPassword.setPreferredSize(new Dimension(50, 20));  
 valueFieldPassword.setMinimumSize(new Dimension(10, 20));  
 valueFieldPassword.setHorizontalAlignment(JTextField.*CENTER*);  
  
 valueFieldPanel.add(valueFieldPassword);  
 add(valueFieldPanel);  
 break;  
 }  
 }  
 }  
  
 public ContentPanel(Color backgroundColor, Color foregroundColor, String title, JButton login, JButton register) {  
 setLayout(new GridLayout(2, 1));  
 setBackground(backgroundColor);  
 setPreferredSize(new Dimension(100, 50));  
 JLabel titleText = new JLabel(title);  
 titleText.setHorizontalAlignment(SwingConstants.*CENTER*);  
 titleText.setForeground(foregroundColor);  
 add(titleText);  
  
 setAlignmentX(Component.*RIGHT\_ALIGNMENT*);  
  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setBackground(backgroundColor);  
 buttonPanel.setLayout(new GridLayout(1, 2));  
 buttonPanel.add(login);  
 buttonPanel.add(register);  
 add(buttonPanel);  
 }  
  
 public ContentPanel(Color backgroundColor, Color foregroundColor, String title, JButton[] btn) {  
 setLayout(new GridLayout(2, 1));  
 setBackground(backgroundColor);  
 setPreferredSize(new Dimension(100, 50));  
 JLabel titleText = new JLabel(title);  
 titleText.setHorizontalAlignment(SwingConstants.*CENTER*);  
 titleText.setForeground(foregroundColor);  
 add(titleText);  
  
 setAlignmentX(Component.*RIGHT\_ALIGNMENT*);  
  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.setBackground(backgroundColor);  
 buttonPanel.setLayout(new GridLayout(1, btn.length));  
  
 for (int i = 0; i < btn.length; i++) {  
 buttonPanel.add(btn[i]);  
 }  
  
 add(buttonPanel);  
 }  
  
 public void setValue(long score) {  
 value.setText("Score: " + Long.*toString*(score));  
 revalidate();  
 repaint();  
 }  
  
 public void setBorder(JPanel obj, int left, int top, int right, int bottom) {  
 Border margin = new EmptyBorder(top, left, bottom, right);  
 obj.setBorder(margin);  
 }  
  
 public String getTextFiled() {  
 return valueField.getText();  
 }  
  
 public String getPassField() {  
 return valueFieldPassword.getText();  
 }  
}

**Window**

package com.NIK;  
  
import com.NIK.Users.CheckableUser;  
import com.NIK.Users.InputUser;  
import com.NIK.Utilities.CheckException;  
import com.NIK.Visual.Canvas;  
import com.NIK.Visual.ContentPanel;  
  
import javax.imageio.ImageIO;  
import javax.swing.\*;  
import javax.swing.border.Border;  
import javax.swing.border.EmptyBorder;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.File;  
import java.io.IOException;  
import java.util.Timer;  
import java.util.TimerTask;  
  
public class Window extends JFrame implements WindowListener {  
 private Database db;  
 private CheckableUser user;  
 private boolean isOp = false;  
  
 private JButton loginButton;  
 private JButton registerButton;  
 private ContentPanel loginPanel;  
 private ContentPanel passwordPanel;  
 private ContentPanel scorePanel;  
  
 private char activeChar = 'A';  
 private JButton charA;  
 private JButton charB;  
 private JButton charC;  
  
 private Timer updateScore;  
 private Timer reachScore;  
 private static int *updateMilliseconds* = 6000;  
 private static int *reachMilliseconds* = 1000;  
  
 public Window() {  
 super("Final Project");  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 setIcon("\\src\\com\\NIK\\assets\\icon.jpg");  
  
 Dimension size = Toolkit.*getDefaultToolkit*().getScreenSize();  
 setPreferredSize(size);  
  
 getContentPane().setBackground(Color.*BLACK*);  
  
 pack();  
 setVisible(true);  
  
 //Event listener  
 addWindowListener(this);  
  
 //Enable databse  
 db = new Database();  
 //!!!! НЕ БОЛЬШЕ 500 !!!!  
 user = new CheckableUser(0, "", "", 0, 20, 20);  
  
 //Every minute update score to server  
 updateScore = new Timer();  
 updateScore.schedule(new TimerTask() {  
 @Override  
 public void run() {  
 user.setScore(user.getScore());  
 db.update(user);  
 }  
 }, 0, *updateMilliseconds*);  
 //Every seconds score  
 reachScore = new Timer();  
 reachScore.schedule(new TimerTask() {  
 @Override  
 public void run() {  
 user.reachScore();  
 if (scorePanel != null)  
 scorePanel.setValue(user.getScore());  
 }  
 }, 0, *reachMilliseconds*);  
  
 //Creating Content---------  
 Container content = getContentPane();  
 content.setLayout(new GridLayout(0, 2));  
  
 Canvas draw = new Canvas(500, 500, user, 20, 20);  
 draw.getImg().addMouseListener(new MouseListener() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
  
 }  
  
 @Override  
 public void mousePressed(MouseEvent e) {  
 if (user.getId() > 0) {  
 if (e.getButton() == MouseEvent.*BUTTON1*) {  
 if (user.getCount() > 0) {  
 draw.setRect(e.getX(), e.getY(), activeChar, user);  
 user.setCount(user.getCount() - 1);  
 charA.setText(Integer.*toString*(user.getCount()));  
 charB.setText(Integer.*toString*(user.getCount()));  
 charC.setText(Integer.*toString*(user.getCount()));  
 } else {  
 System.*out*.println("No many squares!");  
 }  
 } else if (SwingUtilities.*isRightMouseButton*(e)) {  
 if (draw.isRect(e.getX(), e.getY(), user)) {  
 draw.setRect(e.getX(), e.getY(), '\u0001', user);  
 user.setCount(user.getCount() + 1);  
 charA.setText(Integer.*toString*(user.getCount()));  
 charB.setText(Integer.*toString*(user.getCount()));  
 charC.setText(Integer.*toString*(user.getCount()));  
 }  
 }  
 } else if (user.getId() < 0) {  
 System.*out*.println("Ты кто? Ты забанен, блин. И нет я не буду тебя разбавнивать, потому что ты забанин блин");  
 } else {  
 System.*out*.println("Please, sign in or register for play!");  
 }  
 }  
  
 @Override  
 public void mouseReleased(MouseEvent e) {  
  
 }  
  
 @Override  
 public void mouseEntered(MouseEvent e) {  
  
 }  
  
 @Override  
 public void mouseExited(MouseEvent e) {  
  
 }  
 });  
 content.add(draw);  
  
 //Creating active buttons----  
 loginButton = new JButton("Sign in");  
 loginButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 user.setLogin(loginPanel.getTextFiled());  
 user.setPassword(passwordPanel.getPassField());  
 System.*out*.println(db.LoginUser(user, draw));  
 isOp = db.checkOP(user);  
 }  
 });  
 registerButton = new JButton("Sign up");  
 registerButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 System.*out*.println(user.setLoginPro(loginPanel.getTextFiled()));  
 System.*out*.println(user.setPasswordPro(passwordPanel.getPassField()));  
 try {  
 db.registerUser(user, draw);  
 isOp = db.checkOP(user);  
 } catch (CheckException ex) {  
 System.*out*.println(ex);  
 }  
 }  
 });  
 //---------------------------  
  
 GridBagConstraints gbc = new GridBagConstraints();  
  
 Box box = Box.*createVerticalBox*();  
 gbc = new GridBagConstraints();  
 gbc.gridx = 0;  
 gbc.gridy = GridBagConstraints.*RELATIVE*;  
 gbc.weightx = 1.0;  
 gbc.anchor = GridBagConstraints.*FIRST\_LINE\_START*;  
 gbc.fill = GridBagConstraints.*HORIZONTAL*;  
  
 scorePanel = new ContentPanel(Color.*BLACK*, Color.*WHITE*, "Score", 1);  
 box.add(scorePanel, gbc);  
  
 loginPanel = new ContentPanel(Color.*BLACK*, Color.*WHITE*, "Login", 0);  
 box.add(loginPanel, gbc);  
 passwordPanel = new ContentPanel(Color.*BLACK*, Color.*WHITE*, "Password", 3);  
 box.add(passwordPanel, gbc);  
  
 box.add(new ContentPanel(Color.*BLACK*, Color.*WHITE*, "Actions", loginButton, registerButton), gbc);  
  
 //Buttons  
 charA = new JButton("RED");  
 charA.setForeground(Color.*BLACK*);  
 charA.setBackground(Color.*RED*);  
 charA.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 activeChar = 'A';  
 }  
 });  
  
 charB = new JButton("YELLOW");  
 charB.setForeground(Color.*BLACK*);  
 charB.setBackground(Color.*YELLOW*);  
 charB.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 activeChar = 'B';  
 }  
 });  
  
 charC = new JButton("CYAN");  
 charC.setForeground(Color.*BLACK*);  
 charC.setBackground(Color.*CYAN*);  
 charC.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 activeChar = 'C';  
 }  
 });  
  
 JButton[] btns = new JButton[3];  
 btns[0] = charA;  
 btns[1] = charB;  
 btns[2] = charC;  
  
 box.add(new ContentPanel(Color.*BLACK*, Color.*WHITE*, "Square Types", btns), gbc);  
 content.add(box);  
 //-------------------------  
  
 draw.rsize(getWidth(), getHeight());  
 addComponentListener(new ComponentAdapter() {  
 public void componentResized(ComponentEvent componentEvent) {  
 draw.rsize(getWidth(), getHeight());  
 }  
 });  
 }  
  
 public GraphicsConfiguration createGC() {  
 GraphicsEnvironment ge = GraphicsEnvironment.*getLocalGraphicsEnvironment*();  
 GraphicsDevice gs = ge.getDefaultScreenDevice();  
 GraphicsConfiguration gc = gs.getDefaultConfiguration();  
 return gc;  
 }  
  
 public void setIcon(String path) {  
 try {  
 String filePath = new File("").getAbsolutePath();  
 Image icon = ImageIO.*read*(new File(filePath.concat(path)));  
 this.setIconImage(icon);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void setBorder(JLabel obj, int left, int top, int right, int bottom) {  
 Border margin = new EmptyBorder(top,left,bottom,right);  
 obj.setBorder(margin);  
 }  
 public void setBorder(JPanel obj, int left, int top, int right, int bottom) {  
 Border margin = new EmptyBorder(top,left,bottom,right);  
 obj.setBorder(margin);  
 }  
 public void setBorder(JTextField obj, int left, int top, int right, int bottom) {  
 Border margin = new EmptyBorder(top,left,bottom,right);  
 obj.setBorder(margin);  
 }  
  
 @Override  
 public void windowOpened(WindowEvent e) {  
  
 }  
  
 @Override  
 public void windowClosing(WindowEvent e) {  
 db.closeConnection();  
 }  
  
 @Override  
 public void windowClosed(WindowEvent e) {  
 user.setScore(user.getScore());  
 db.update(user);  
 }  
  
 @Override  
 public void windowIconified(WindowEvent e) {  
  
 }  
  
 @Override  
 public void windowDeiconified(WindowEvent e) {  
  
 }  
  
 @Override  
 public void windowActivated(WindowEvent e) {  
  
 }  
  
 @Override  
 public void windowDeactivated(WindowEvent e) {  
  
 }  
  
 public String banUser(String login) {  
 if (isOp) {  
 db.banUser(login);  
 return "Ban function for " + login + " is applied!";  
 } else {  
 return "Not enough rights for this command!";  
 }  
 }  
}

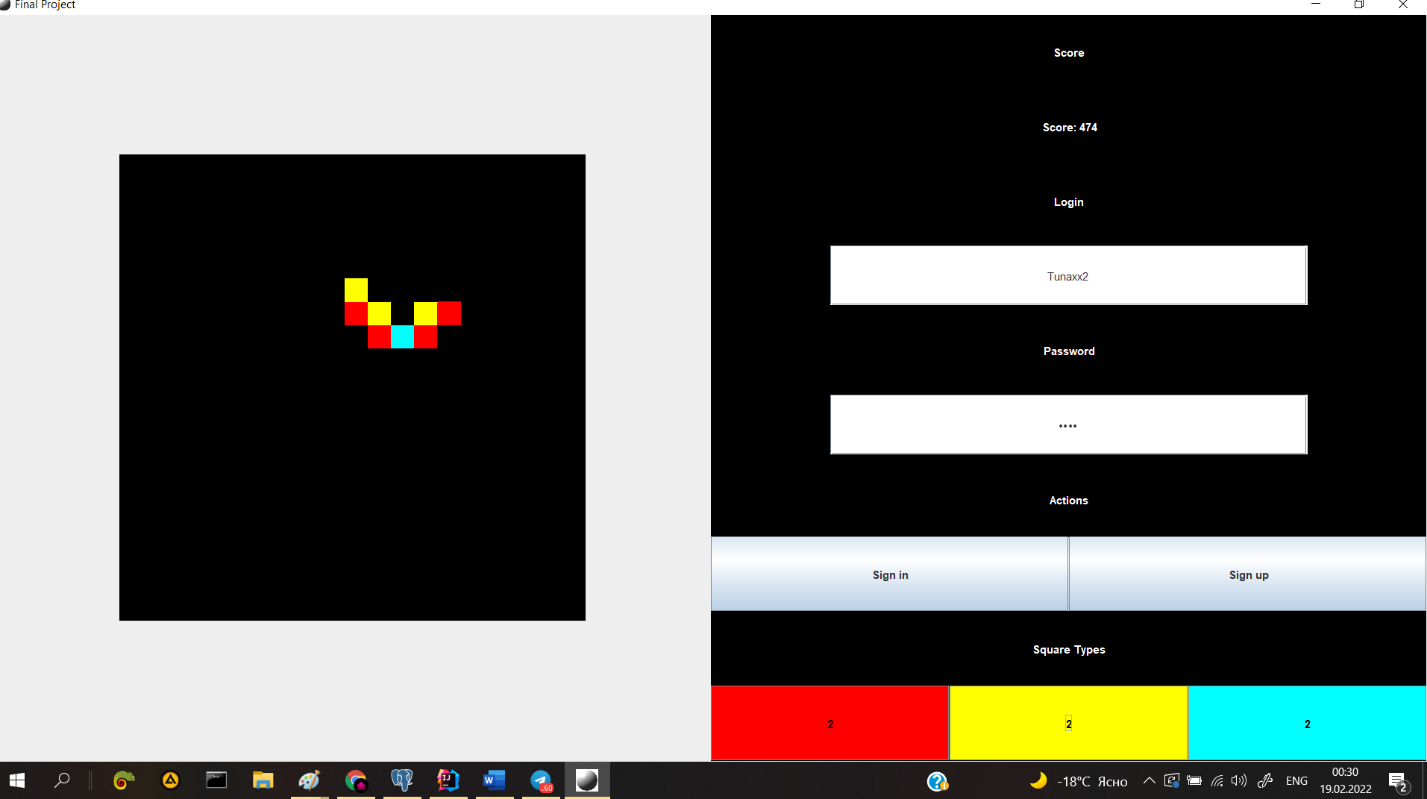
**Database**

package com.NIK;  
  
import com.NIK.Users.CheckableUser;  
import com.NIK.Users.InputUser;  
import com.NIK.Utilities.CheckException;  
import com.NIK.Visual.Canvas;  
  
import java.sql.\*;  
  
public class Database {  
 private Connection con = null;  
 private Statement st = null;  
  
 public Database() {  
 try {  
 Class.*forName*("org.postgresql.Driver");  
 con = DriverManager.*getConnection*("jdbc:postgresql://localhost:5432/postgres", "postgres", "tunaxx");  
 if (con != null) {  
 System.*out*.println("Connection successful! " + con.getCatalog());  
 }  
 } catch(ClassNotFoundException e) {  
 e.printStackTrace();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
  
 public void closeConnection() {  
 try {  
 if (con != null) con.close();  
 if (st != null) st.close();  
 System.*out*.println("Connection closed!");  
 } catch (SQLException e) {}  
 }  
  
 public void update(CheckableUser user) {  
 if (user.getId() != 0) {  
 String state = "UPDATE finalproject SET score = " + user.getScore() + " WHERE id = '" + user.getId() + "'";  
 try {  
 st = con.createStatement();  
 st.executeUpdate(state);  
 st.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 state = "UPDATE finalproject SET map = '" + String.*valueOf*(user.getBits()) + "' WHERE id = '" + user.getId() + "'";  
 try {  
 st = con.createStatement();  
 st.executeUpdate(state);  
 st.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 state = "UPDATE finalproject SET count = '" + String.*valueOf*(user.getCount()) + "' WHERE id = '" + user.getId() + "'";  
 try {  
 st = con.createStatement();  
 st.executeUpdate(state);  
 st.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void registerUser(CheckableUser user, Canvas draw) throws CheckException {  
 if (user.getLogin().equals("") || user.getLogin() == null) throw new CheckException("No login!");  
 if (user.getPassword().equals("") || user.getPassword() == null) throw new CheckException("No password!");  
  
 if (isLoginExist(user.getLogin())) throw new CheckException("This login is already exist!");  
  
 try {  
 int id = calculateNextId();  
 if (id == 0) throw new SQLException("SQL connection failed!");  
 String state = "INSERT INTO finalproject VALUES (";  
 state += Integer.*toString*(id) + ", '";  
 state += user.getLogin() + "', '";  
 char[] bits = user.getBits();  
 state += user.getPassword() + "', 0, '";  
 for (int i = 0; i < bits.length; i++) {  
 state += bits[i];  
 }  
 state += "',";  
 state += Integer.*toString*(user.getWidth()) + ", ";  
 state += Integer.*toString*(user.getHeight()) + ", 10)";  
 System.*out*.println(user.setIdPro(id));  
  
 draw.rsize(user);  
 user.setCount(10);  
  
 st = con.createStatement();  
 st.executeUpdate(state);  
 st.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public boolean checkOP(CheckableUser user) {  
 PreparedStatement prepStmt = null;  
 try {  
 prepStmt = con.prepareStatement("SELECT OPLogin FROM finalprojectops", ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);  
 ResultSet rs = prepStmt.executeQuery();  
 while(rs.next()) {  
 if (rs.getString("oplogin").equals(user.getLogin())) return true;  
 }  
 rs.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return false;  
 }  
  
 public String LoginUser(CheckableUser user, Canvas draw) {  
 if (!isPasswordCorrect(user.getLogin(), user.getPassword())) return "Password is incorrect!";  
  
 user.setScore(user.getScore());  
 update(user);  
  
 PreparedStatement prepStmt = null;  
 try {  
 prepStmt = con.prepareStatement("SELECT \* FROM finalproject WHERE LOGIN = '" + user.getLogin() + "'", ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);  
 ResultSet rs = prepStmt.executeQuery();  
 rs.next();  
  
 user.setBits(Integer.*parseInt*(rs.getString("width")), Integer.*parseInt*(rs.getString("height")));  
  
 user.setBits(rs.getString("map"));  
 System.*out*.println(user.setIdPro(Integer.*parseInt*(rs.getString("id"))));  
 draw.rsize((InputUser)user);  
 draw.bitMap(user.getBits(), user);  
 user.setScore(Integer.*parseInt*(rs.getString("score")));  
 user.setCount(Integer.*parseInt*(rs.getString("count")));  
 rs.close();  
 return "Successful sign in!";  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "";  
 }  
  
 public boolean isLoginExist(String login) {  
 PreparedStatement prepStmt = null;  
 try {  
 prepStmt = con.prepareStatement("SELECT LOGIN FROM finalproject", ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);  
 ResultSet rs = prepStmt.executeQuery();  
 while(rs.next()) {  
 if (rs.getString("login").equals(login)) return true;  
 }  
 rs.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return false;  
 }  
  
 public boolean isPasswordCorrect(String login, String password) {  
 try {  
 PreparedStatement prepStmt = null;  
 String state = "SELECT password FROM finalproject WHERE login = '" + login + "'";  
 prepStmt = con.prepareStatement(state, ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);  
 ResultSet rs = prepStmt.executeQuery();  
 rs.next();  
 if (rs.getString("password").equals(password)) {  
 return true;  
 } else {  
 return false;  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return true;  
 }  
  
 public void banUser(String login) {  
 String state = "UPDATE finalproject SET id = -id WHERE login = '" + login + "'";  
 try {  
 st = con.createStatement();  
 st.executeUpdate(state);  
 st.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 //If returns 0 - SQL error  
 public int calculateNextId() {  
 PreparedStatement prepStmt = null;  
 try {  
 prepStmt = con.prepareStatement("SELECT ID FROM finalproject", ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);  
 ResultSet rs = prepStmt.executeQuery();  
  
 int rowCount = 1;  
 while(rs.next()) {  
 rowCount++;  
 }  
 rs.close();  
 return rowCount;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return 0;  
 }  
}

**Main**

package com.NIK;  
  
import com.NIK.Users.CheckableUser;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.util.Scanner;  
import java.util.Timer;  
import java.util.TimerTask;  
  
public class Main {  
 private static Window *window*;  
 public static void main(String[] args) {  
 //Running implementation thread in Swing JFrame classes  
 //Main window  
 javax.swing.SwingUtilities.*invokeLater*(new Runnable() {  
 public void run() {  
 *window* = new Window();  
 }  
 });  
  
 Timer cmd = new Timer();  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
 cmd.schedule(new TimerTask() {  
 @Override  
 public void run() {  
 String command = null;  
 try {  
 command = reader.readLine();  
 System.*out*.println(*checkCommand*(command));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }, 0, 1000);  
 }  
  
 public static String checkCommand(String command) {  
 if (command.substring(0, 5).equals("/ban ")) {  
 return *window*.banUser(command.substring(5, command.length()));  
 }  
 return "This command is not found!";  
 }  
}

**Screenshots:**

****

Изображение выглядит как текст, монитор, снимок экрана, электроника

Автоматически созданное описание

**Conclusion:**

In this project and in good Java practice course I learned a lot about OOP concepts. Got clear explanations from professor. Tried to practice my knowledge in this project; I found some improvements and flaws in my work during this development time: In OOP code should always be structured, up to the order of the methods inside classes; If this long project, I need to leave comments not under the announcement of function, but under the declaration; User interfaces and abstract classes more often when I don’t want to declare many objects inside. Use annotations more often and learn how to do it; It's better to practice with the new library in a new project; Do not create function on the task, but create class for all varieties and everything connected with this task. I want to express my gratitude for PhD. Aitmukhanbetova Elvira for competent and qualified instructions. I can confidently say that I have improved my knowledge in OOP, Java, and JDBC.