**ליאור סברי - 313137937**

**בן ביטון – 203330378**

**Read before running the program:**

* There is a .jpg file in the SRC folder so please run the .java files from the same location.
* Two new buttons have been added 1st is showListButton which shows has the clubbers in the array list in any given state, 2nd is deleteButton which lets us delete an existing clubber in a similar way we search a clubber.
* It is possible to enable the closure of a new ClubAbstractEntity window because the data will only be saved into the array list only after the creation window is closed and the data will also first be checked if its valid before addition, but I didn’t want to take the risk and lose points.
* Once a creation window is opened it is not possible to touch the main JFrame since in my opinion it is a better way to operate.

**Reason for warning in NightClubMgmtApp line 252 –**

// SuppressWarnings doesn't work because of a bug, therefore it isn't

// possible to ignore the warning. The warning happens because we read

// an object from a file which obviously can't be known beforehand and

// then try to cast it to an ArrayList<ClubAbstractEntity>, even if we

// check with 'instanceof' the warning will presist, so the only option

// we have is to leave it as is.

//@SuppressWarnings("unchecked")

clubbers = (ArrayList<ClubAbstractEntity>)ois.readObject();

**Class ClubAbstractEntity – ClubAbstractEntity.java**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.io.\*;

/\*\*

\* This abstract class extends from {@link javax.swing.JFrame} and implements

\* {@link java.io.Serializable}.

\* <br> This Class builds the basic GUI arragment of each window.

\* <br> Other classes that inherit from this class may add {@link java.awt.Component}s

\* to the centerPanel which is set to {@link java.awt.GridLayout}.

\* <br> This class holds the inner class {@link ButtonsHandler} which listens

\* to the {@link javax.swing.JButton} okButton and cancelButton.

\* <br> Currently the class that inherits from this class is: {@link Person}

\* @author Lior Sabri, Ben Biton

\*/

public abstract class ClubAbstractEntity extends JFrame implements Serializable

{

private JButton okButton;

private JButton cancelButton;

private JPanel centerPanel;

// It's a good practice to save a reference to events/listeners that are in

// the memory to prevent memory leak.

private ButtonsHandler handler;

/\*\*

\* ClubAbstractEntity Constructor - initializes JButtons, JPanels and

\* {@link ButtonsHandler}.

\*/

public ClubAbstractEntity()

{

// Setting the GridLayout with (0, 1) will make it add a new row when needed.

centerPanel = new JPanel(new GridLayout(0, 1));

JPanel buttonPanel = new JPanel();

okButton = new JButton("OK");

cancelButton = new JButton("Cancel");

cancelButton.setEnabled(false);

handler = new ButtonsHandler();

okButton.addActionListener(handler);

cancelButton.addActionListener(handler);

buttonPanel.add(okButton);

buttonPanel.add(cancelButton);

add(centerPanel);

add(buttonPanel, BorderLayout.SOUTH);

setVisible(true);

setResizable(false);

setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

}

/\*\*

\* This method recieves a {@link java.awt.Component} and adds it to the CenterPanel

\* @param guiComponent holds a component sent by son classes.

\*/

protected void addToCenter(Component guiComponent)

{

centerPanel.add(guiComponent);

}

/\*\*

\* Abstract method that implements a copmarison with a given key.

\* @param key String which will be compared.

\* @return true or false whether the key has found a match.

\*/

public abstract boolean match(String key);

/\*\*

\* Abstract method which invokes whenever okay button is pressed and

\* implements data validation.

\* @return true or false whether the data is valid.

\*/

protected abstract boolean validateData();

/\*\*

\* Abstract method which invokes whenever okayButton is pressed if

\* {@link #validateData} returns true.

\*/

protected abstract void commit();

/\*\*

\* Abstract method which invokes whenever cancelButton is pressed.

\*/

protected abstract void rollBack();

/\*\*

\* Abstract method which helps know if fields of inherited class are initialized.

\* @return true or false whether the data is empty.

\*/

public abstract boolean isEmpty();

/\*\*

\* Abstract method which returns inherited class ID.

\* <br> This method is only needed in {@link Person}, although if it will be

\* placed there then when used in {@link NightClubMgmtApp} {@link ClubAbstractEntity}

\* will need casting to {@link Person}.

\* @return recieve an ID.

\*/

public abstract String getID();

/\*\*

\* This Class implemets {@link java.awt.event.ActionListener} and

\* {@link java.io.Serializable}.

\* <br> ButtonsHandler handles okButton and cancelButton.

\*/

private class ButtonsHandler implements ActionListener ,Serializable

{

/\*\*

\* This method is invoked when okButton and cancelButton are pressed.

\* <br> okButton - first checks {@link #validateData} if it returns true

\* it will proceed to {@link #commit}, enable cancel button with

\* {@link java.awt.Component#setEnabled} and hide the {@link javax.swing.JFrame}.

\* <br> cancelButton - invokes {@link #rollBack} and then sets

\* {@link javax.swing.JFrame} visibility to false.

\* @param event Holds the current event.

\*/

public void actionPerformed(ActionEvent event)

{

Object source = event.getSource();

if (source == okButton)

{

if (validateData())

{

commit();

cancelButton.setEnabled(true);

setVisible(false);

}

return;

}

if (source == cancelButton)

{

rollBack();

setVisible(false);

}

}

}

}

**Class Person – Person.java**

import javax.swing.\*;

import java.awt.\*;

/\*\*

\* This class extends from {@link ClubAbstractEntity}.

\* <br> This Class sets info fields for a person with GUI elements which then

\* sends them to {@link ClubAbstractEntity} through

\* {@link ClubAbstractEntity#addToCenter}.

\* This class implements the abrstract methods: {@link ClubAbstractEntity#match},

\* {@link ClubAbstractEntity#validateData}, {@link ClubAbstractEntity#commit},

\* {@link ClubAbstractEntity#rollBack}, {@link ClubAbstractEntity#isEmpty} and

\* {@link ClubAbstractEntity#getID}.

\* @author Lior Sabri, Ben Biton

\*/

public class Person extends ClubAbstractEntity

{

private String id;

private String name;

private String surname;

private String tel;

private JTextField idTextField;

private JTextField nameTextField;

private JTextField surnameTextField;

private JTextField telTextField;

private JLabel[] asteriskLabel;

/\*\*

\* Person Empty Constructor - invokes arguments constructor with empty strings.

\*/

public Person()

{

this("", "", "", "");

}

/\*\*

\* Person Arguments Constructor - creates an info {@link javax.swing.JPanel}

\* with {@link java.awt.FlowLayout} for each info field and sends them to

\* {@link ClubAbstractEntity#addToCenter}.

\* <br> Initializes all {@link java.lang.String}s, {@link javax.swing.JTextField}s

\* and {@link javax.swing.JLabel}s and add to them a tooltip with the method

\* {@link javax.swing.JComponent#setToolTipText}.

\* @param id persons ID.

\* @param name persons name.

\* @param surname persons surname.

\* @param tel persons phone number.

\*/

public Person(String id, String name, String surname, String tel)

{

JPanel idPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JPanel namePanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JPanel surnamePanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

JPanel telPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

idTextField = new JTextField(30);

nameTextField = new JTextField(30);

surnameTextField = new JTextField(30);

telTextField = new JTextField(30);

idTextField.setText(id);

nameTextField.setText(name);

surnameTextField.setText(surname);

telTextField.setText(tel);

this.id = idTextField.getText();

this.name = idTextField.getText();

this.surname = idTextField.getText();

this.tel = idTextField.getText();

JLabel idLabel = new JLabel("ID");

JLabel nameLabel = new JLabel("Name");

JLabel surnameLabel = new JLabel("Surname");

JLabel telLabel = new JLabel("Tel");

idLabel.setToolTipText("e.g: 0-2423535|1, 2-5554445|3");

nameLabel.setToolTipText("e.g: Mark, Bo");

surnameLabel.setToolTipText("e.g: Avrahami-O'Mally, Sabri");

telLabel.setToolTipText("e.g: +(972)50-6663210, +(44)206-8208167, +(1)4-9520205");

asteriskLabel = new JLabel[4];

JPanel[] asteriskPanel = new JPanel[4];

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

{

asteriskLabel[i] = new JLabel("\*");

asteriskLabel[i].setForeground(Color.RED);

asteriskLabel[i].setVisible(false);

// Reason for the JPanel is because in Java it isn't possible to hide

// and when you set visibility to false the JLabel will collapse and

// create a less desirable GUI.

asteriskPanel[i] = new JPanel();

asteriskPanel[i].setPreferredSize(new Dimension(15, 25));

asteriskPanel[i].add(asteriskLabel[i]);

}

idPanel.add(idLabel);

idPanel.add(idTextField);

idPanel.add(asteriskPanel[0]);

namePanel.add(nameLabel);

namePanel.add(nameTextField);

namePanel.add(asteriskPanel[1]);

surnamePanel.add(surnameLabel);

surnamePanel.add(surnameTextField);

surnamePanel.add(asteriskPanel[2]);

telPanel.add(telLabel);

telPanel.add(telTextField);

telPanel.add(asteriskPanel[3]);

addToCenter(idPanel);

addToCenter(namePanel);

addToCenter(surnamePanel);

addToCenter(telPanel);

setTitle("Person Clubber's Data");

setSize(450, 220);

}

/\*\*

\* This method returns the {@link Person}s ID.

\* @return id {@link java.lang.String}

\*/

public String getID()

{

return id;

}

/\*\*

\* This method checks whether the info fields are empty with

\* {@link java.lang.Object#equals}.

\* @return true or false whether an info field is empty.

\*/

public boolean isEmpty()

{

// Comparing an empty String is an efficient way to both check whether

// the String is empty or null, since equals() will check its arguement and

// return false incase it is null.

String emptyString = "";

return emptyString.equals(id) || emptyString.equals(name) ||

emptyString.equals(surname) || emptyString.equals(tel);

}

/\*\*

\* This method checks whether the key sent is equal with

\* {@link java.lang.Object#equals} to id.

\* @param key may hold an id String.

\* @return true or false whether id is equal to the key.

\*/

public boolean match(String key)

{

return id.equals(key);

}

/\*\*

\* This method validates whether the info fields matches regular expressions

\* with {@link java.lang.String#matches} if so it will set the asteriskLabels

\* visibility to true, otherwise to false.

\* @return true or false whether all info fields match thier regular expressions.

\*/

protected boolean validateData()

{

if (!idTextField.getText().matches("\\d-\\d{7}\\|[1-9]"))

{

asteriskLabel[0].setVisible(true);

return false;

}

else

{

asteriskLabel[0].setVisible(false);

}

if (!nameTextField.getText().matches("[A-Z][a-z]+"))

{

asteriskLabel[1].setVisible(true);

return false;

}

else

{

asteriskLabel[1].setVisible(false);

}

if (!surnameTextField.getText().matches("([A-Z][a-z]\*['-]?)+"))

{

asteriskLabel[2].setVisible(true);

return false;

}

else

{

asteriskLabel[2].setVisible(false);

}

if (!telTextField.getText().matches("\\+\\([1-9]\\d{0,2}\\)[1-9]\\d{0,2}-[1-9]\\d{6}"))

{

asteriskLabel[3].setVisible(true);

return false;

}

else

{

asteriskLabel[3].setVisible(false);

}

return true;

}

/\*\*

\* This method uses {@link javax.swing.JTextField#getText} to recieve the text

\* in the {@link javax.swing.JTextField}s and save them in the info Strings.

\*/

protected void commit()

{

id = idTextField.getText();

name = nameTextField.getText();

surname = surnameTextField.getText();

tel = telTextField.getText();

}

/\*\*

\* This method uses {@link javax.swing.JTextField#setText} to insert the info

\* String into the {@link javax.swing.JTextField}s and set the asteriskLabels

\* visibility to false.

\*/

protected void rollBack()

{

idTextField.setText(id);

nameTextField.setText(name);

surnameTextField.setText(surname);

telTextField.setText(tel);

for (JLabel asterisk : asteriskLabel)

{

asterisk.setVisible(false);

}

}

/\*\*

\* Overriden method to implement {@link Person}'s {@link java.lang.Object#toString}.

\*/

@Override

public String toString()

{

return String.format("ID: %s\nName: %s\nSurname: %s\nTel: %s\n",

id, name, surname, tel);

}

}

**Class Soldier – Soldier.java**

import javax.swing.\*;

import java.awt.\*;

/\*\*

\* This class extends from {@link Person}.

\* <br> This Class sets info fields for a person with GUI elements which then

\* sends them to {@link ClubAbstractEntity} through

\* {@link ClubAbstractEntity#addToCenter}.

\* This class overrides the methods: {@link Person#match},

\* {@link Person#validateData}, {@link Person#commit},

\* {@link Person#rollBack} and {@link Person#isEmpty}.

\* @author Lior Sabri, Ben Biton

\*/

public class Soldier extends Person

{

private String personalNumber;

private JTextField personalNumberTextField;

private JLabel asteriskLabel;

/\*\*

\* Soldier Empty Constructor - invokes arguments constructor with empty strings.

\*/

public Soldier()

{

this("", "", "", "", "");

}

/\*\*

\* Soldier Arguments Constructor - creates an info {@link javax.swing.JPanel}

\* with {@link java.awt.FlowLayout} for info field and sends them it

\* {@link ClubAbstractEntity#addToCenter}.

\* <br> Initializes {@link java.lang.String}, {@link javax.swing.JTextField}

\* and {@link javax.swing.JLabel}s and add to them a tooltip with the method

\* {@link javax.swing.JComponent#setToolTipText}.

\* @param id soldiers ID.

\* @param name soldiers name.

\* @param surname soldiers surname.

\* @param tel soldiers phone number.

\* @param personalNumber soldiers personal number.

\*/

public Soldier(String id, String name, String surname, String tel,

String personalNumber)

{

super(id, name, surname, tel);

personalNumberTextField = new JTextField(30);

personalNumberTextField.setText(personalNumber);

this.personalNumber = personalNumberTextField.getText();

JLabel personalNumberLabel = new JLabel("Personal No.");

personalNumberLabel.setToolTipText("e.g: R/4684109, O/5044109, C/4684109");

asteriskLabel = new JLabel("\*");

asteriskLabel.setForeground(Color.RED);

asteriskLabel.setVisible(false);

JPanel asteriskPanel = new JPanel();

asteriskPanel.setPreferredSize(new Dimension(15, 25));

asteriskPanel.add(asteriskLabel);

JPanel personalNumberPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

personalNumberPanel.add(personalNumberLabel);

personalNumberPanel.add(personalNumberTextField);

personalNumberPanel.add(asteriskPanel);

addToCenter(personalNumberPanel);

setTitle("Soldier Clubber's Data");

setSize(450, 250);

}

/\*\*

\* This method checks whether the personalNumber is empty with

\* {@link java.lang.Object#equals} or the Super method {@link Person#isEmpty}.

\* @return true or false whether an info field is empty.

\*/

@Override

public boolean isEmpty()

{

return super.isEmpty() || "".equals(personalNumber);

}

/\*\*

\* This method checks whether the key sent is equal with

\* {@link java.lang.Object#equals} to personalNumber or to the Super method

\* {@link Person#match}.

\* @param key may hold an id or personalNumber String.

\* @return true or false whether id, personalNumber or both are equal to the key.

\*/

@Override

public boolean match(String key)

{

return super.match(key) || personalNumber.equals(key);

}

/\*\*

\* This method validates whether the info field matches regular expressions

\* with {@link java.lang.String#matches} if so it will set the asteriskLabel

\* visibility to true, otherwise to false.

\* <br> It also invokes the Super method {@link Person#validateData}.

\* @return true or false whether all info fields match thier regular expressions.

\*/

@Override

protected boolean validateData()

{

if (!super.validateData())

{

return false;

}

if (!personalNumberTextField.getText().matches("[ROC]/[1-9]\\d{6}"))

{

asteriskLabel.setVisible(true);

return false;

}

else

{

asteriskLabel.setVisible(false);

}

return true;

}

/\*\*

\* This method uses {@link javax.swing.JTextField#getText} to recieve the text

\* in the {@link javax.swing.JTextField} and save them in the info String.

\* <br> It also invokes the Super method {@link Person#commit}.

\*/

@Override

protected void commit()

{

super.commit();

personalNumber = personalNumberTextField.getText();

}

/\*\*

\* This method uses {@link javax.swing.JTextField#setText} to insert the info

\* String into the {@link javax.swing.JTextField} and set the asteriskLabel

\* visibility to false.

\* <br> It also invokes the Super method {@link Person#rollBack}.

\*/

@Override

protected void rollBack()

{

super.rollBack();

personalNumberTextField.setText(personalNumber);

asteriskLabel.setVisible(false);

}

/\*\*

\* Overriden method to implement {@link Soldier} {@link java.lang.Object#toString}.

\* <br> It also invokes the Super method {@link Person#toString}.

\*/

@Override

public String toString()

{

return super.toString() + String.format("Personal Number: %s\n", personalNumber);

}

}

**Class Student – Student.java**

import javax.swing.\*;

import java.awt.\*;

/\*\*

\* This class extends from {@link Person}.

\* <br> This Class sets info fields for a person with GUI elements which then

\* sends them to {@link ClubAbstractEntity} through

\* {@link ClubAbstractEntity#addToCenter}.

\* This class overrides the methods: {@link Person#match},

\* {@link Person#validateData}, {@link Person#commit},

\* {@link Person#rollBack} and {@link Person#isEmpty}.

\* @author Lior Sabri, Ben Biton

\*/

public class Student extends Person

{

private String studentID;

private JTextField studentIDTextField;

private JLabel asteriskLabel;

/\*\*

\* Student Empty Constructor - invokes arguments constructor with empty strings.

\*/

public Student()

{

this("", "", "", "", "");

}

/\*\*

\* Student Arguments Constructor - creates an info {@link javax.swing.JPanel}

\* with {@link java.awt.FlowLayout} for info field and sends them it

\* {@link ClubAbstractEntity#addToCenter}.

\* <br> Initializes {@link java.lang.String}, {@link javax.swing.JTextField}

\* and {@link javax.swing.JLabel}s and add to them a tooltip with the method

\* {@link javax.swing.JComponent#setToolTipText}.

\* @param id student ID.

\* @param name student name.

\* @param surname student surname.

\* @param tel student phone number.

\* @param studentID student ID.

\*/

public Student(String id, String name, String surname, String tel,

String studentID)

{

super(id, name, surname, tel);

studentIDTextField = new JTextField(30);

studentIDTextField.setText(studentID);

this.studentID = studentIDTextField.getText();

JLabel studentIDLabel = new JLabel("Student ID");

studentIDLabel.setToolTipText("e.g: SCE12345, RUP56098, HIT14606");

asteriskLabel = new JLabel("\*");

asteriskLabel.setForeground(Color.RED);

asteriskLabel.setVisible(false);

JPanel asteriskPanel = new JPanel();

asteriskPanel.add(asteriskLabel);

asteriskPanel.setPreferredSize(new Dimension(15, 25));

JPanel studentIDPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

studentIDPanel.add(studentIDLabel);

studentIDPanel.add(studentIDTextField);

studentIDPanel.add(asteriskPanel);

addToCenter(studentIDPanel);

setTitle("Student Clubber's Data");

setSize(450, 250);

}

/\*\*

\* This method checks whether the studentID is empty with

\* {@link java.lang.Object#equals} or the Super method {@link Person#isEmpty}.

\* @return true or false whether an info field is empty.

\*/

@Override

public boolean isEmpty()

{

return super.isEmpty() || studentID.equals("");

}

/\*\*

\* This method checks whether the key sent is equal with

\* {@link java.lang.Object#equals} to studentID from the 4th letter

\* (including) or to the Super method {@link Person#match}.

\* @param key may hold an id or numbers of studentID String.

\* @return true or false whether id, studentID or both are equal to the key.

\*/

@Override

public boolean match(String key)

{

return super.match(key) || key.regionMatches(0, studentID, 3, 5);

}

/\*\*

\* This method validates whether the info field matches regular expressions

\* with {@link java.lang.String#matches} if so it will set the asteriskLabel

\* visibility to true, otherwise to false.

\* <br> It also invokes the Super method {@link Person#validateData}.

\* @return true or false whether all info fields match thier regular expressions.

\*/

@Override

protected boolean validateData()

{

if (!super.validateData())

{

return false;

}

if (!studentIDTextField.getText().matches("[A-Z]{3}[1-9]\\d{4}"))

{

asteriskLabel.setVisible(true);

return false;

}

else

{

asteriskLabel.setVisible(false);

}

return true;

}

/\*\*

\* This method uses {@link javax.swing.JTextField#getText} to recieve the text

\* in the {@link javax.swing.JTextField} and save them in the info String.

\* <br> It also invokes the Super method {@link Person#commit}.

\*/

@Override

protected void commit()

{

super.commit();

studentID = studentIDTextField.getText();

}

/\*\*

\* This method uses {@link javax.swing.JTextField#setText} to insert the info

\* String into the {@link javax.swing.JTextField} and set the asteriskLabel

\* visibility to false.

\* <br> It also invokes the Super method {@link Person#rollBack}.

\*/

@Override

protected void rollBack()

{

super.rollBack();

studentIDTextField.setText(studentID);

asteriskLabel.setVisible(false);

}

/\*\*

\* Overriden method to implement {@link Student} {@link java.lang.Object#toString}.

\* <br> It also invokes the Super method {@link Person#toString}.

\*/

@Override

public String toString()

{

return super.toString() + String.format("Student ID: %s\n", studentID);

}

}

**Class NightClubMgmtApp – NightClubMgmtApp.java**

// File: NightClubMgmtApp.java

import java.util.\*;

import javax.swing.\*;

import javax.swing.border.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

/\*\*

\* This abstract class extends from {@link javax.swing.JFrame}.

\* The class provides a user interface which will be able to communicate in a

\* user friendly manner.

\* <br> This Class holds an {@link java.util.ArrayList}&lt;{@link ClubAbstractEntity}&gt;

\* which provides a window with {@link javax.swing.JTextField} so the user can write

\* the clubbers info.

\* <br> This class holds the inner class {@link ButtonsHandler} which listens

\* to the {@link javax.swing.JButton} searchButton, showListButton, clubberButton

\* and deleteButton.

\* <br> The {@link javax.swing.JRadioButton} personRadioButton, soldierRadioButton

\* and studentRadioButton are set to a button group which provide the user a way

\* to choose what type of clubber they would like to create.

\* <br> The class has an instance called overlayPanel which is a

\* {@link ImagePanel} that is set to a {@link java.awt.GridLayout}.

\* <br> There is an instance called self which holds the 'this' of the class

\* which will be helpful to window placements.

\* @author Lior Sabri, Ben Biton

\*/

public class NightClubMgmtApp extends JFrame

{

// Night-Club Regular Customer Repository.

private ArrayList<ClubAbstractEntity> clubbers;

// JButtons which provide the same functionality as their name.

private JButton searchButton;

private JButton showListButton;

private JButton clubberButton;

private JButton deleteButton;

// JRadioButtons which help choose the type of the clubber.

private JRadioButton personRadioButton;

private JRadioButton soldierRadioButton;

private JRadioButton studentRadioButton;

// Main JPanel that holds other GUI elements.

private ImagePanel overlayPanel;

// Holds the 'this' of the class.

private JFrame self;

// It's a good practice to save a reference to events/listeners that are in

// the memory to prevent memory leak.

private ButtonsHandler btnHandler;

/\*\*

\* NightClubMgmtApp Constructor - initializes the {@link javax.swing.JFrame}

\* and places it in the center of the screen with

\* {@link java.awt.Window#setLocationRelativeTo} with sending null, when null

\* is sent location will be set to the center of the screen.

\* <br> Initializes {@link javax.swing.JRadioButton} into a

\* {@link javax.swing.ButtonGroup} with the method {@link #createButtonPanel}.

\* <br> Initializes {@link javax.swing.JButton} and {@link ButtonsHandler}

\* with the method {@link #createButtonPanel}.

\* <br> Initializes the {@link java.util.ArrayList}&lt;{@link ClubAbstractEntity}&gt;

\* and read from file with method {@link #loadClubbersDBFromFile} and when

\* closing the program there is a {@link java.awt.event.WindowListener} which

\* Overrides the method {@link java.awt.event.WindowAdapter#windowClosing}

\* which when invoked it will write the clubbers data to a file with the

\* method {@link #writeClubbersDBtoFile}.

\*/

public NightClubMgmtApp()

{

super("Night Club Management App");

self = this;

// Create an image icon and get the background image.

overlayPanel = new ImagePanel(new ImageIcon(getClass().

getResource("background.jpg")).getImage());

Color skinColor = new Color(170, 105, 100);

Color greyBlue = new Color(125, 145, 180);

// Create a LineBorder and TitleBorder for a nice looking menu.

LineBorder bkLineBorder = new LineBorder(greyBlue, 1);

TitledBorder bkTitledBorder = new TitledBorder(bkLineBorder, "B.K",

TitledBorder.CENTER, TitledBorder.DEFAULT\_POSITION,

new Font(Font.SERIF, Font.BOLD + Font.ITALIC, 40), greyBlue);

overlayPanel.setBorder(bkTitledBorder);

add(overlayPanel);

// Add an underline to the JLabel and set the font.

JLabel radioTitleLabel = new JLabel("<HTML><U>Clubber Type:</U></HTML>",

SwingConstants.CENTER);

radioTitleLabel.setFont(new Font(Font.SERIF, Font.BOLD + Font.ITALIC, 20));

radioTitleLabel.setForeground(skinColor);

overlayPanel.add(radioTitleLabel);

createRadioButtonPanel();

// Add an underline to the JLabel and set the font.

JLabel buttonsTitleLabel = new JLabel("<HTML><U>Clubbers Controls:</U></HTML>",

SwingConstants.CENTER);

buttonsTitleLabel.setFont(new Font(Font.SERIF, Font.BOLD + Font.ITALIC, 30));

buttonsTitleLabel.setForeground(skinColor);

overlayPanel.add(buttonsTitleLabel);

createButtonPanel();

// Get size of screen and set width and hieght to make the program

// appear in a fitting size to the screen.

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

int height = screenSize.height;

int width = screenSize.width;

// Set size of JFrame and location to the center of the screen.

setSize(width/4, height/4);

setLocationRelativeTo(null);

setVisible(true);

setResizable(false);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

clubbers = new ArrayList<>();

loadClubbersDBFromFile();

addWindowListener(new WindowAdapter()

{

/\*\*

\* Overriden method which invokes the method

\* {@link #writeClubbersDBtoFile} while closing the window.

\* @param event Holds the current event.

\*/

@Override

public void windowClosing(WindowEvent event)

{

writeClubbersDBtoFile();

System.exit(0);

}

});

}

/\*\*

\* This method Initializes the {@link javax.swing.JButton}, sets their color,

\* sets a hand cursor when hovering the buttons with the method

\* {@link java.awt.Component#setCursor}, sets a tooltip with the method

\* {@link javax.swing.JComponent#setToolTipText} and adds an

\* {@link java.awt.event.ActionListener} to {@link ButtonsHandler}.

\* <br> A {@link javax.swing.JPanel} that is set to a {@link java.awt.GridLayout}

\* will set its opaque to false for better design with the method

\* {@link javax.swing.JComponent#setOpaque}.

\*/

private void createButtonPanel()

{

JPanel buttonPanel = new JPanel(new GridLayout(2, 2, 5, 5));

buttonPanel.setOpaque(false);

btnHandler = new ButtonsHandler();

searchButton = new JButton("Search");

showListButton = new JButton("Show List");

clubberButton = new JButton("New Clubber");

deleteButton = new JButton("Delete Clubber");

Color darkCyan = new Color(125, 145, 180);

searchButton.setBackground(darkCyan);

showListButton.setBackground(darkCyan);

clubberButton.setBackground(darkCyan);

deleteButton.setBackground(darkCyan);

searchButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

showListButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

clubberButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

deleteButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

searchButton.setToolTipText(

"Search for a clubber using an ID, personal number or student ID");

showListButton.setToolTipText("Show all saved clubbers details");

clubberButton.setToolTipText("Choose clubber type with radio buttons");

deleteButton.setToolTipText(

"Delete a clubber using an ID, personal number or student ID");

searchButton.addActionListener(btnHandler);

showListButton.addActionListener(btnHandler);

clubberButton.addActionListener(btnHandler);

deleteButton.addActionListener(btnHandler);

buttonPanel.add(searchButton);

buttonPanel.add(showListButton);

buttonPanel.add(clubberButton);

buttonPanel.add(deleteButton);

overlayPanel.add(buttonPanel);

}

/\*\*

\* This method Initializes the {@link javax.swing.JRadioButton}, sets a

\* hand cursor when hovering the JRadioButtons with the method

\* {@link java.awt.Component#setCursor} and adds them to a

\* {@link javax.swing.ButtonGroup}

\* <br> A {@link javax.swing.JPanel} will set its opaque to false for better

\* design with the method

\* {@link javax.swing.JComponent#setOpaque}.

\*/

private void createRadioButtonPanel()

{

JPanel radioButtonPanel = new JPanel();

radioButtonPanel.setOpaque(false);

personRadioButton = new JRadioButton("Person", true);

soldierRadioButton = new JRadioButton("Soldier", false);

studentRadioButton = new JRadioButton("Student", false);

Color greyBlue = new Color(70, 100, 125);

personRadioButton.setForeground(greyBlue);

soldierRadioButton.setForeground(greyBlue);

studentRadioButton.setForeground(greyBlue);

personRadioButton.setOpaque(false);

soldierRadioButton.setOpaque(false);

studentRadioButton.setOpaque(false);

personRadioButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

soldierRadioButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

studentRadioButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

radioButtonPanel.add(personRadioButton);

radioButtonPanel.add(soldierRadioButton);

radioButtonPanel.add(studentRadioButton);

ButtonGroup radioGroup = new ButtonGroup();

radioGroup.add(personRadioButton);

radioGroup.add(soldierRadioButton);

radioGroup.add(studentRadioButton);

overlayPanel.add(radioButtonPanel);

}

/\*\*

\* The method loads the clubbers data from a file. If no file is found a

\* message will appear

\* <br> See FileOutputStream class: {@link java.io.FileInputStream}.

\* <br> See ObjectOutputStream class: {@link java.io.ObjectInputStream}.

\* <br> See writeObject method: {@link java.io.ObjectInputStream#readObject}.

\*/

private void loadClubbersDBFromFile()

{

FileInputStream fis = null;

ObjectInputStream ois = null;

try

{

// Open file.

fis = new FileInputStream("BKCustomers.dat");

ois = new ObjectInputStream(fis);

// Read object from a file.

// SuppressWarnings doesn't work because of a bug, therefore it isn't

// possible to ignore the warning. The warning happens because we read

// an object from a file which obviously can't be known beforehand and

// then try to cast it to an ArrayList<ClubAbstractEntity>, even if we

// check with 'instanceof' the warning will presist, so the only option

// we have is to leave it as is.

//@SuppressWarnings("unchecked")

clubbers = (ArrayList<ClubAbstractEntity>)ois.readObject();

}

catch (FileNotFoundException e)

{

JOptionPane.showMessageDialog(this,

"BKCustomers.dat will be created after closing the program.",

"File Not Found", JOptionPane.INFORMATION\_MESSAGE);

}

catch (IOException e)

{

}

catch (ClassNotFoundException e)

{

}

finally // Close inputstreams if they exist.

{

try

{

if (ois != null)

{

ois.close();

}

if (fis != null)

{

fis.close();

}

}

catch (IOException e)

{

}

}

}

/\*\*

\* The method saves the current clubbers into file.

\* <br> See FileOutputStream class: {@link java.io.FileOutputStream}.

\* <br> See ObjectOutputStream class: {@link java.io.ObjectOutputStream}.

\* <br> See writeObject method: {@link java.io.ObjectOutputStream#writeObject}.

\*/

private void writeClubbersDBtoFile()

{

FileOutputStream fos = null;

ObjectOutputStream oos = null;

try

{

// Create a new file.

fos = new FileOutputStream("BKCustomers.dat");

oos = new ObjectOutputStream(fos);

// Write object to a file

oos.writeObject(clubbers);

oos.flush();

}

catch (FileNotFoundException e)

{

}

catch (IOException e)

{

}

finally // Close outputstreams if they exist

{

try

{

if (oos != null)

{

oos.close();

}

if (fos != null)

{

fos.close();

}

}

catch (IOException e)

{

}

}

}

/\*\*

\* This clas extends from {@link javax.swing.JPanel}.

\* <br> This class is a useful way to set a background image to a JPanel.

\*/

private class ImagePanel extends JPanel

{

private Image image = null;

private int iWidth2;

private int iHeight2;

/\*\*

\* ImagePanel Constructor recieves a {@link java.awt.Image} initializes

\* its class instance variables with image height and length.

\* @param image holds an image to set as background.

\*/

public ImagePanel(Image image)

{

setLayout(new GridLayout(0, 1));

this.image = image;

iWidth2 = image.getWidth(this)/2;

iHeight2 = image.getHeight(this)/2;

}

/\*\*

\* The method will draw the image to the JPanel using the method

\* {@link java.awt.Graphics#drawImage}.

\* @param g The graphic panel.

\*/

public void paintComponent(Graphics g)

{

super.paintComponent(g);

if (image != null)

{

int x = this.getParent().getWidth()/2 - iWidth2;

int y = this.getParent().getHeight()/2 - iHeight2;

g.drawImage(image, 0, 0, getWidth(), getHeight(), this);

}

}

}

/\*\*

\* This Class implemets {@link java.awt.event.ActionListener}.

\* <br> ButtonsHandler listens to the {@link javax.swing.JButton}:

\* searchButton, showListButton, clubberButton and deleteButton.

\*/

private class ButtonsHandler implements ActionListener

{

/\*\*

\* This method is invoked when searchButton, showListButton,

\* clubberButton or deleteButton are pressed.

\* <br> The buttons will invoke the following methods:

\* <br> searchButton - {@link #searchClubber}.

\* <br> showListButton - {@link #showClubbers}.

\* <br> clubberButton - {@link #createClubber}.

\* <br> deleteButton - {@link #deleteClubber}.

\* @param event Holds the current event.

\*/

public void actionPerformed(ActionEvent event)

{

Object source = event.getSource();

if (source == searchButton)

{

String userInput = JOptionPane.showInputDialog(self,

"Please Enter The Clubber's Key:", "Seach Key",

JOptionPane.QUESTION\_MESSAGE);

searchClubber(userInput);

return;

}

if (source == showListButton)

{

showClubbers();

return;

}

if (source == clubberButton)

{

createClubber();

return;

}

if (source == deleteButton)

{

String userInput = JOptionPane.showInputDialog(self,

"Please Enter The Clubber's Key:", "Delete Key",

JOptionPane.QUESTION\_MESSAGE);

deleteClubber(userInput);

return;

}

}

/\*\*

\* This method recieves a key and runs through they arraylist to find

\* a match with the method {@link ClubAbstractEntity#match}, if found

\* the entity will set its visibility to true and allow customization

\* and prevent use of the main JFrame until the current frame is closed.

\* @param key holds data to find in the arraylist.

\*/

private void searchClubber(String key)

{

if (key == null)

{

return;

}

for (ClubAbstractEntity clubber : clubbers)

{

if (clubber.match(key))

{

// Disable use of main frame.

self.setEnabled(false);

// Set the clubber window to appear in the middle

// of the frame.

clubber.setLocationRelativeTo(self);

clubber.setAlwaysOnTop(true);

clubber.setVisible(true);

clubber.addWindowListener(new WindowAdapter()

{

/\*\*

\* Overriden method which will prevent access to the main

\* frame till the current window is closed.

\* @param event Holds the current event.

\*/

@Override

public void windowDeactivated(WindowEvent event)

{

// As long as the window is visible don't proceed

if (clubber.isVisible())

{

return;

}

// Enable use of main frame.

self.setEnabled(true);

// Stop listening to this window after it isn't visible

clubber.removeWindowListener(this);

}

});

return;

}

}

JOptionPane.showMessageDialog(self,

String.format("Clubber with key %s does not exist", key),

"Key Not Found", JOptionPane.INFORMATION\_MESSAGE);

}

/\*\*

\* This method will show all the clubbers in a {@link javax.swing.JTextArea}

\* and add a {@link javax.swing.JScrollPane} to it.

\*/

private void showClubbers()

{

String clubbersDetails = String.format("Number of Clubbers: %d\n\n",

clubbers.size());

// Add all clubbers into one string

for (ClubAbstractEntity clubber : clubbers)

{

clubbersDetails += clubber + "\n";

}

JTextArea textArea = new JTextArea(clubbersDetails);

JScrollPane scrollPane = new JScrollPane(textArea);

textArea.setLineWrap(true);

textArea.setWrapStyleWord(true);

scrollPane.setPreferredSize(new Dimension(0, 350));

JOptionPane.showMessageDialog(self, scrollPane, "Clubbers Details",

JOptionPane.INFORMATION\_MESSAGE);

}

/\*\*

\* This method enables the creation of the objects {@link Person},

\* {@link Soldier} and {@link Student} through their empty constructors

\* they will be added to the arraylist only after the window visibility

\* will be set to false with help of a {@link java.awt.event.WindowListener}

\* and the ID doesn't already exist in the arraylist.

\*/

private void createClubber()

{

// Prevent use of main frame.

self.setEnabled(false);

ClubAbstractEntity newClubber;

if (personRadioButton.isSelected())

{

newClubber = new Person();

}

else if (soldierRadioButton.isSelected())

{

newClubber = new Soldier();

}

else

{

newClubber = new Student();

}

// Set the new clubber window to appear in the middle

// of the frame

newClubber.setLocationRelativeTo(self);

newClubber.addWindowListener(new WindowAdapter()

{

/\*\*

\* Overriden method which will prevent access to the main

\* frame till the current window has been closed.

\* Upon closing the window the method will then check if the new

\* clubber is valid with the methods: {@link ClubAbstractEntity#match}

\* and {@link ClubAbstractEntity#getID}

\* @param event Holds the current event.

\*/

@Override

public void windowDeactivated(WindowEvent event)

{

// As long as the window is visible don't continue

if (newClubber.isVisible())

{

return;

}

// Enable use of main frame.

self.setEnabled(true);

// Stop listening to this window after it isn't visible

newClubber.removeWindowListener(this);

// Prevent addition of empty clubbers.

// This happens when the program is force closed.

if (newClubber.isEmpty())

{

return;

}

// Check if there is a clubber with the same ID

for (ClubAbstractEntity clubber : clubbers)

{

if (clubber.match(newClubber.getID()))

{

JOptionPane.showMessageDialog(self,

"ID already exists", "Clubber Addition Error",

JOptionPane.ERROR\_MESSAGE);

return;

}

}

clubbers.add(newClubber);

}

});

}

/\*\*

\* This method gives the user an option to delete a clubber with a key

\* which holds data.

\* @param key holds data to find in the arraylist.

\*/

private void deleteClubber(String key)

{

if (key == null)

{

return;

}

for (ClubAbstractEntity clubber : clubbers)

{

if (clubber.match(key))

{

int inputYesNo = JOptionPane.showConfirmDialog(self,

String.format("Are you sure you want to delete:\n%s",clubber),

"Delete Warning", JOptionPane.YES\_NO\_OPTION,

JOptionPane.WARNING\_MESSAGE);

if (inputYesNo == JOptionPane.YES\_OPTION)

{

clubbers.remove(clubber);

JOptionPane.showMessageDialog(self,

"Clubber succesfuly deleted!",

"Clubber Deleted", JOptionPane.INFORMATION\_MESSAGE);

}

return;

}

}

JOptionPane.showMessageDialog(self,

String.format("Clubber with key %s does not exist", key),

"Key Not Found", JOptionPane.INFORMATION\_MESSAGE);

}

}

/\*\*

\* static main method.

\* Simply invokes {@link #NightClubMgmtApp} constructor.

\* @param args command line arguments

\*/

public static void main(String[] args)

{

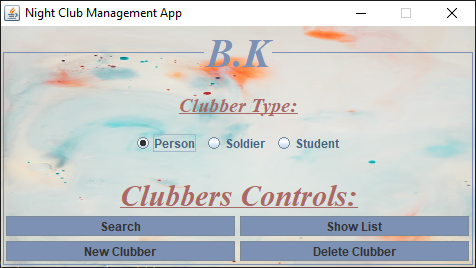
NightClubMgmtApp application = new NightClubMgmtApp();

}

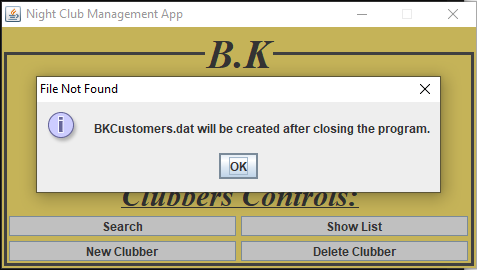
} // End of class NightClubMgmtApp

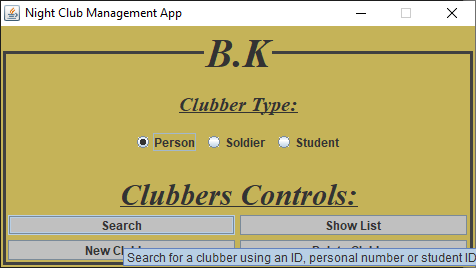
**Print Screen**

**New Design:**

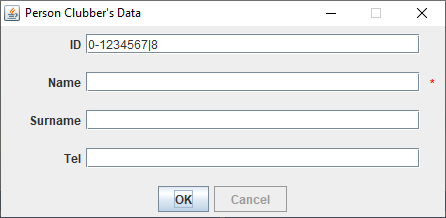


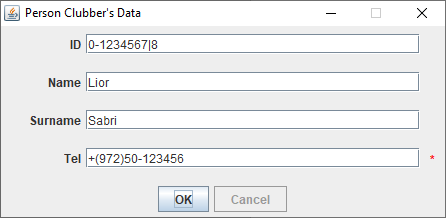
**Old design:**

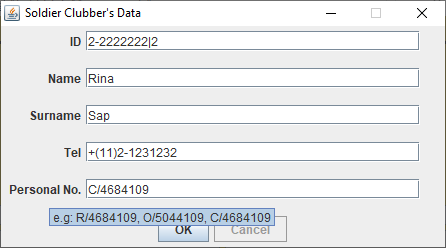
****

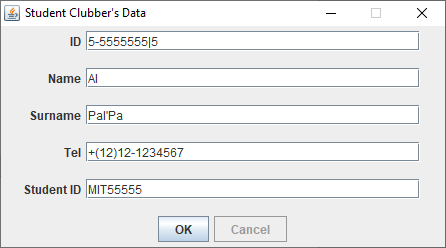


**New Clubber button example:**

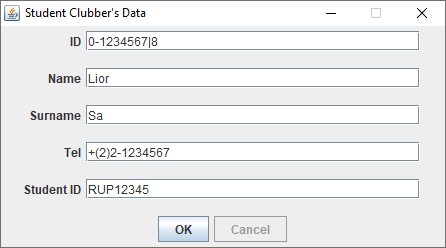


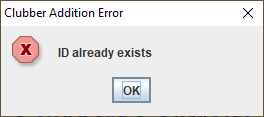




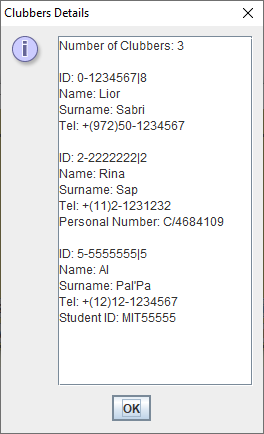


**Same ID example:**

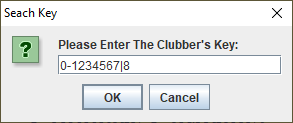


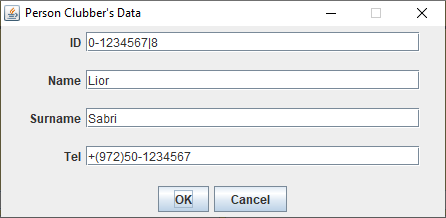


**ShowList button example:**



**Search button example:**





**Dletete Clubber button example:**

