

Required External Documentation

**Title:** Family Tree App

**Author:** Mike Gomes Camara

**Date:** 31/05/2016

**File names:** familytreeApp.java, family1.txt, family2.txt, family3.txt, family4.txt (different versions of the same file)

**Purpose**: Read a txt file, display a family tree and gives a GUI interface that allows, editing and creation of new people and families.

**Requirements/Specification:**

The program is a GUI java app that allows user to load a txt file and display the people listed in the file as several families that are displayed individually and also in a family tree structure.

The user has to have the means to edit persons or just visualize them. Also, the user should be able to insert new families, and add people to the existent families.

The file has to be a .TXT File that each line contain all details about a person, all details are separated by a “;” and new person is delimited by new lines.

The output of the program is a single person from a particular family choosen being displayed at the time, and the user is able to navigate among the members of a family, as well as to see the entire family being displayed in a family tree structure.

The algorithm is based on the proper insertion of information by the user, in the child level field the user is suppose to insert only a number from 0 to is not a child, from one to n if is the first or subsequent child.

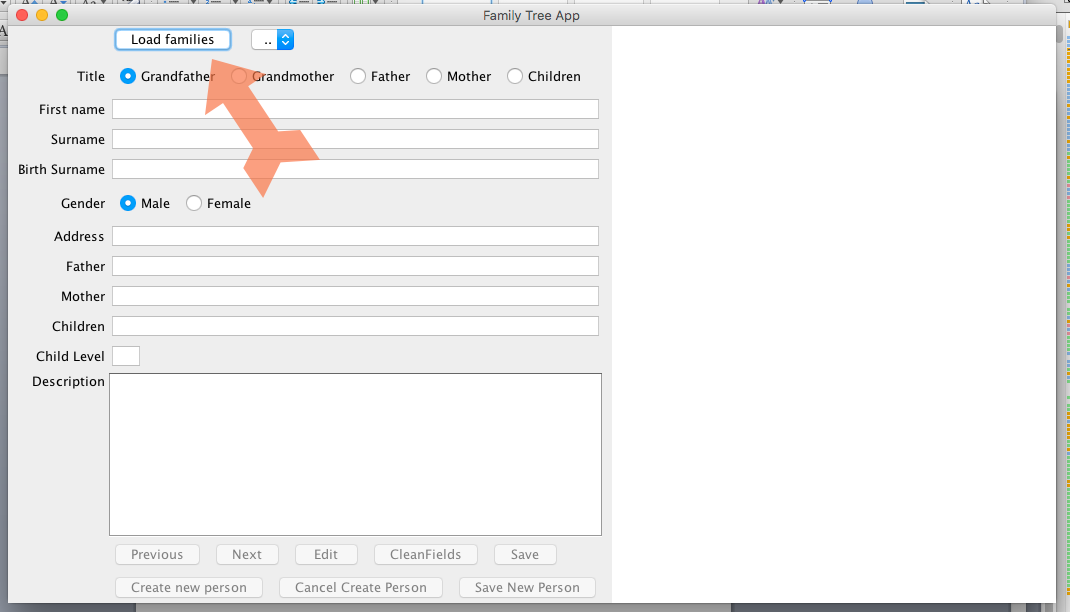
**User Guide:** instructions on how to compile, run and use the program.

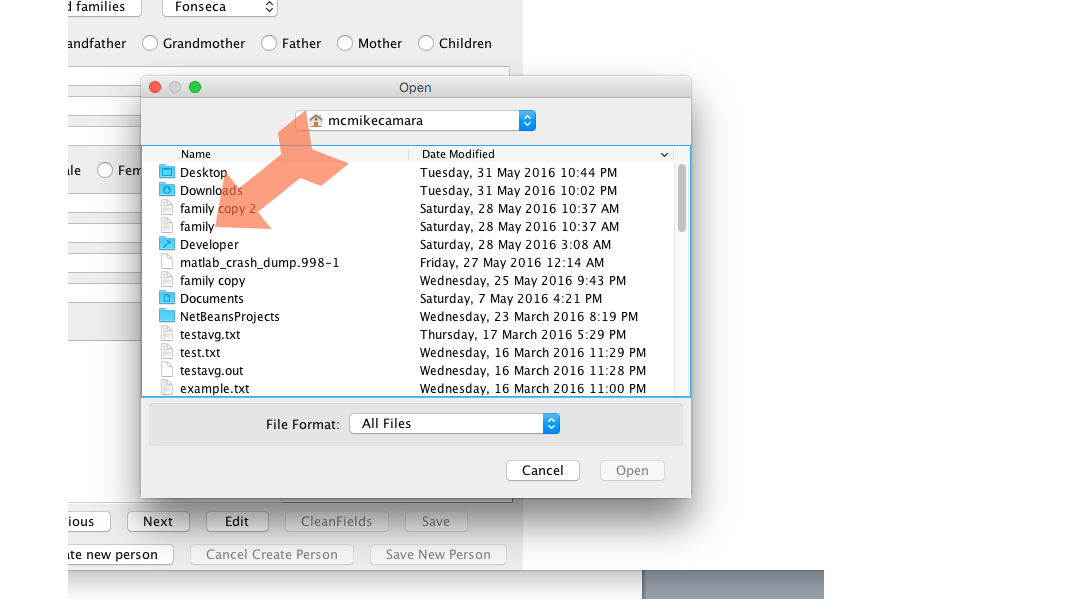
In order to run the program the user have first to open net beans and open the fileproject FamilyTreeApp provided.

1 – Open app on Netbeans

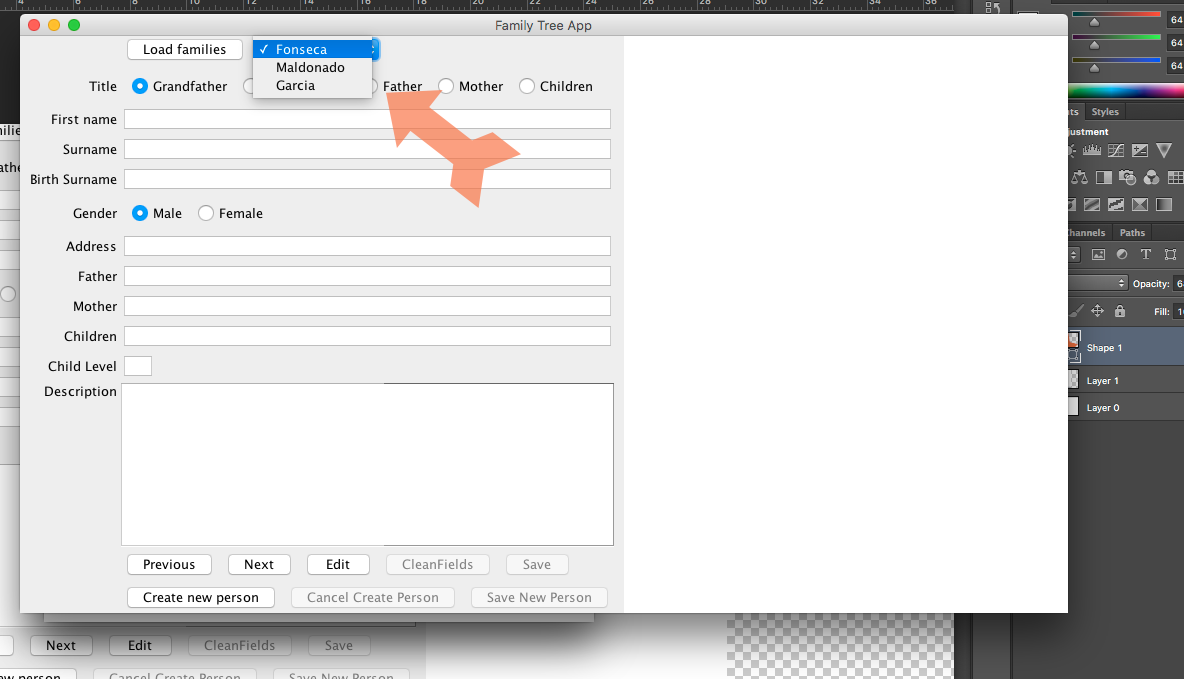
2 – Run (compile) the program on Netbeans

3 – Load families, by clicking the button “Load families” and choosing the the file family.txt

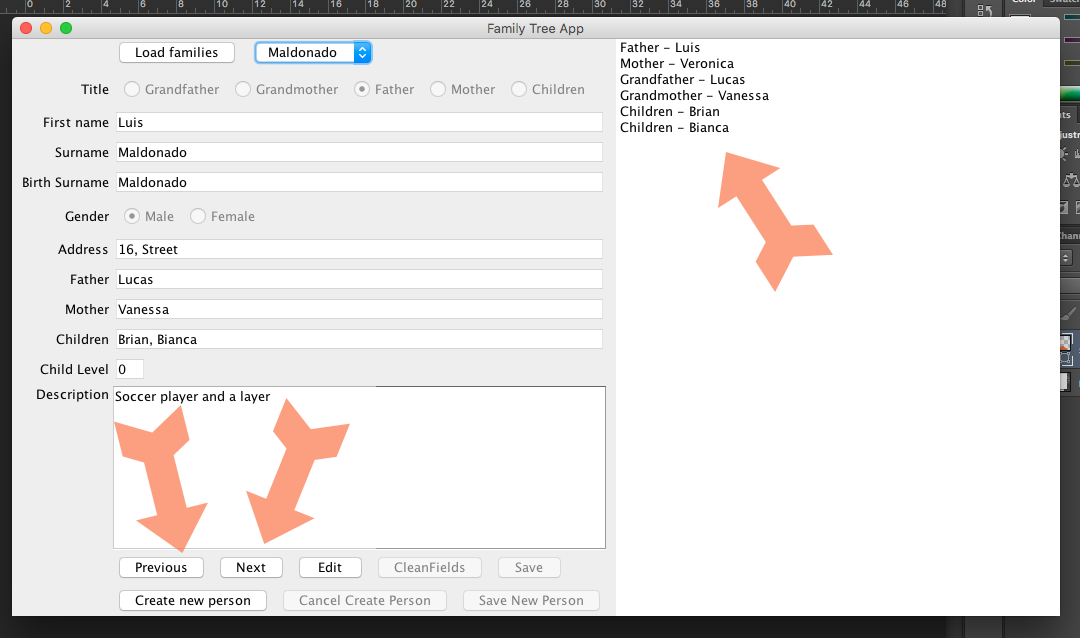




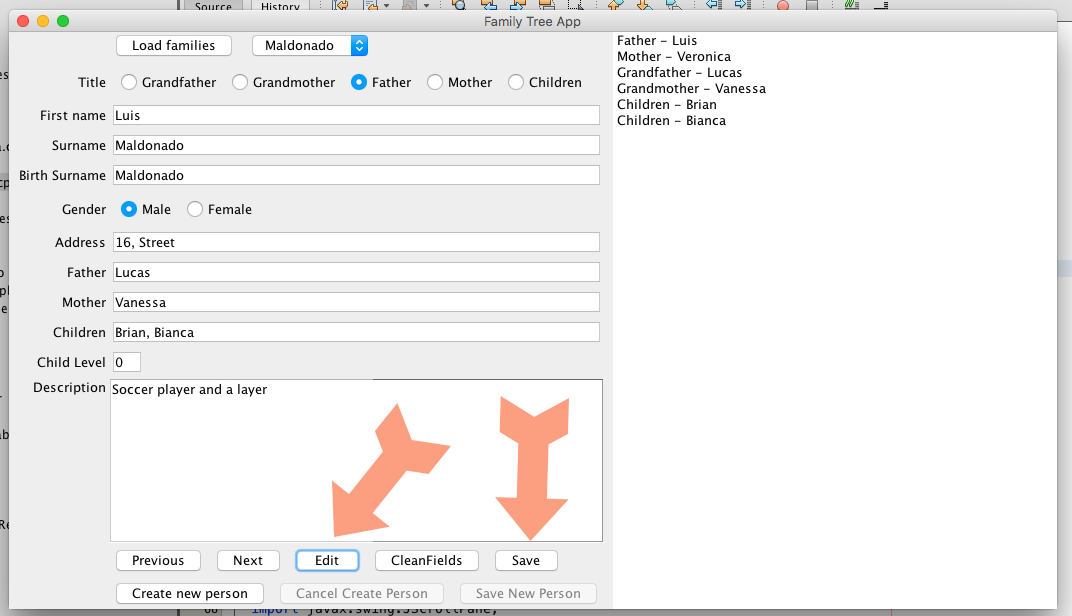
4- Once you load the families choose one family from the list



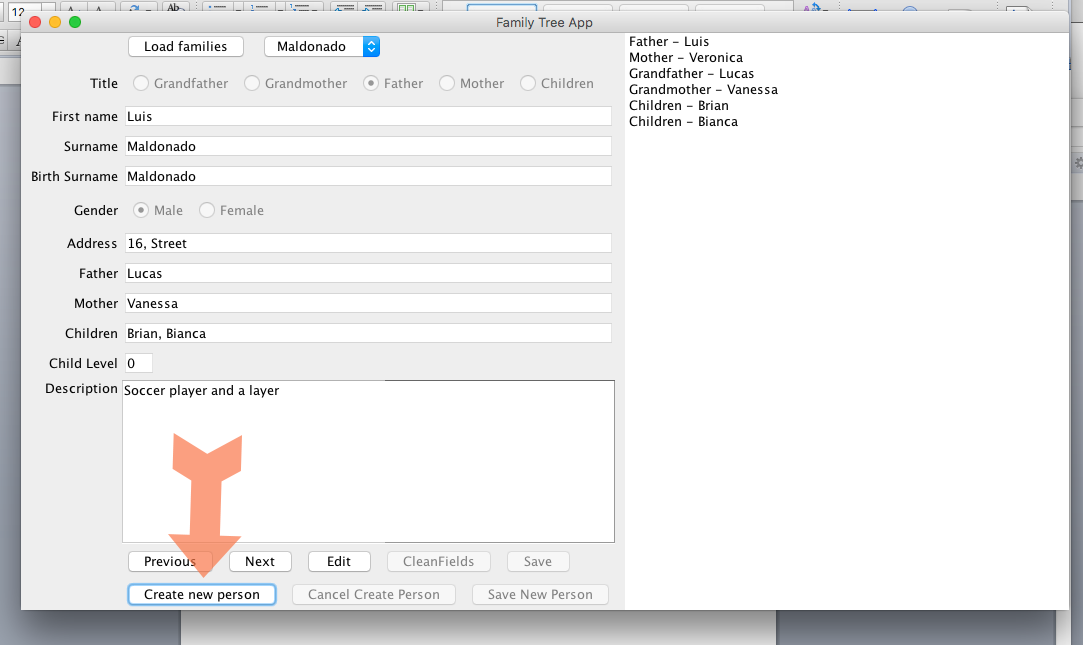
5- After selecting one of the families available you can navigate among the members of the family by pressing the Previous and Next button, as well as check the entire family tree in the right hand side of the user interface.



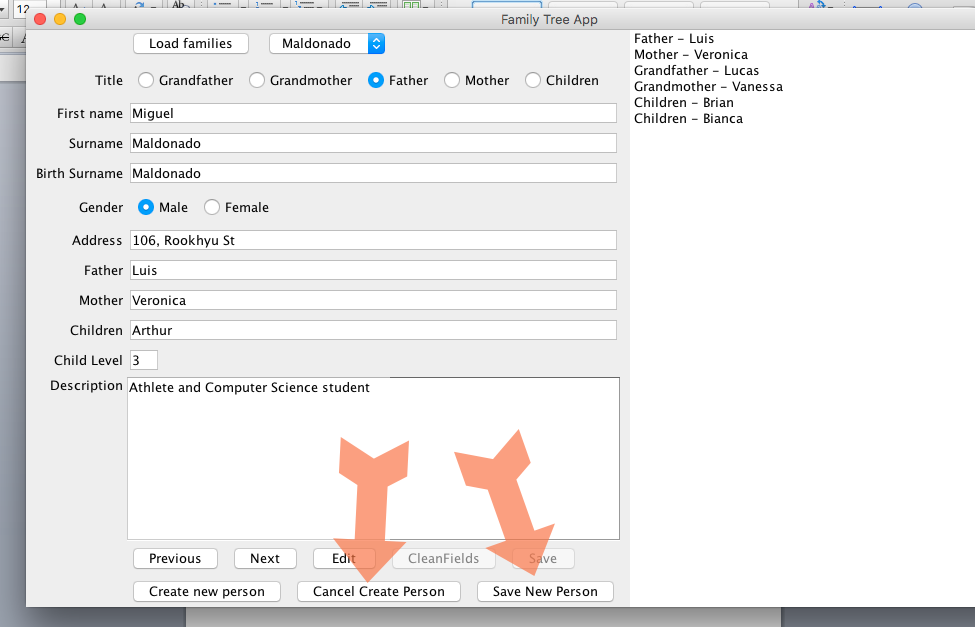
6- To Edit one of the person displayed you would have to click on the “Edit button” and the fields will be enabled for edition, then you press save and the new information edited will be stored in the file .txt.



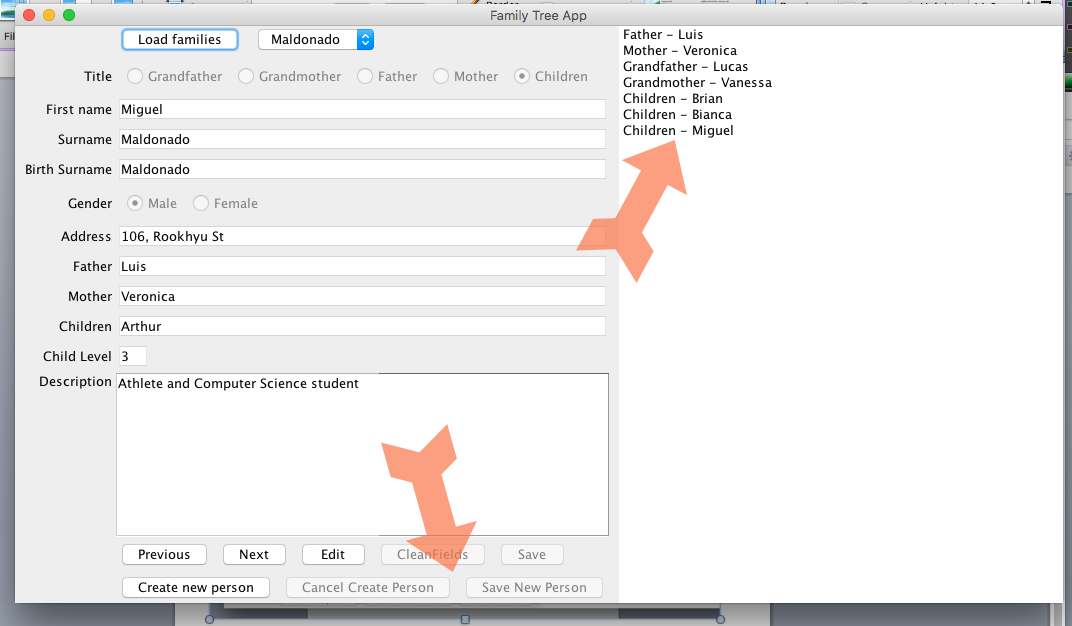
7 – To add a new Person to the family as well as to add new families, you would to click on the “create new person button”



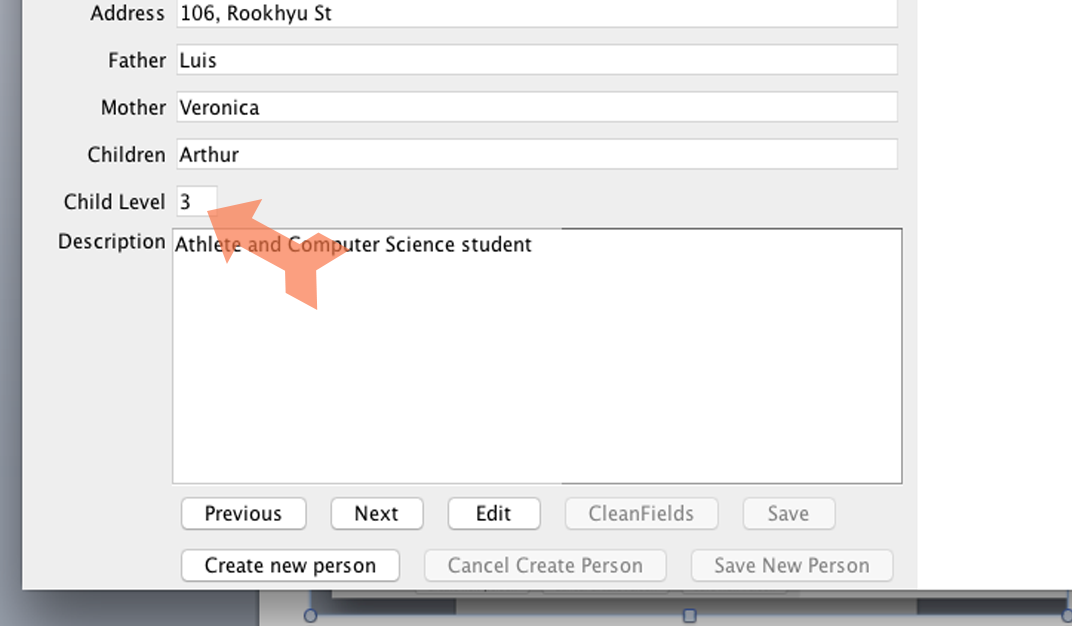
8 – After you click on it, the fields will empty up and you will be able to add the new person. After you type all information of the new person, you can either cancel the operation or save the new person in the file.



9 – If you click in the button “save new person”, the algorithm will check by the surname if it is a new family or an existent family, if it belongs to an existent surname it will add the new person in the Family tree. Otherwise will add the new person to a new family, which you will be able to add people.



10 – It is important to notice that the algorithm relay on the correct insertion of child level in the textfield, for example, it is the first born the child level would be 1, and so on.



**Structure/Design:**

**Design:**

As Data Structure I stored all objects of person in a HashMap, that would work like a dictionary with keys and values, I decided for this approach, but towards the end of the project I realize that I should have done differently. If I had more time, I would modularize and split this file in different files. Also, I would use different classes, such as Person, Address and more instead of saving all objects as a Hashmap, and I created an ArrayList of Hashmaps.

Also, in terms of algorithm logic, in my iterations I used brute force, most of the time looping all objects of the ArrayList to check something, however, if I had more time, I would consider storing each family in a different ArrayList, reducing considerably the amount of time to process them all.

Despite it runs okay now, I realize that with the approach that I have chosen, it would give me trouble in case I was reading a massive file, with thousands of lines, probably would slow everything down, because the amount of iterators that I have used.

The reason that I choose this inefficient approach was my lack of familiarity with the Java Platform, but especially my lack of confidence with GUI interfaces, when I finally got the GUI working, I barely had time to finish the app itself.

So at the moment I have one single class for the entire program, a class called:

public class FamilyTreeApp

The methods in this class are:

FamilyTreeApp()

selectFile()

disableFields()

cleanFields()

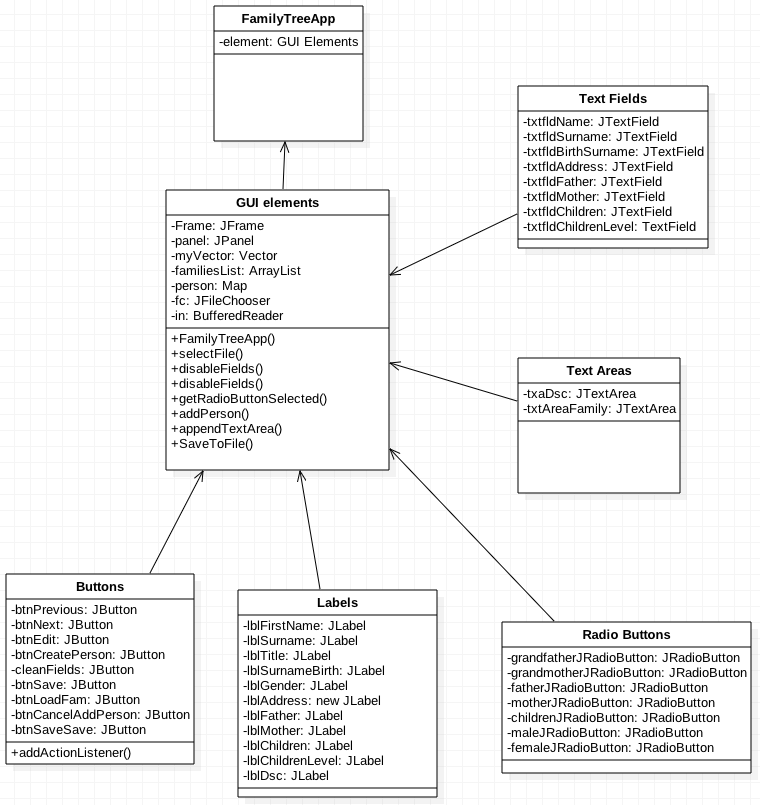
getRadioButtonSelected()

addPerson()

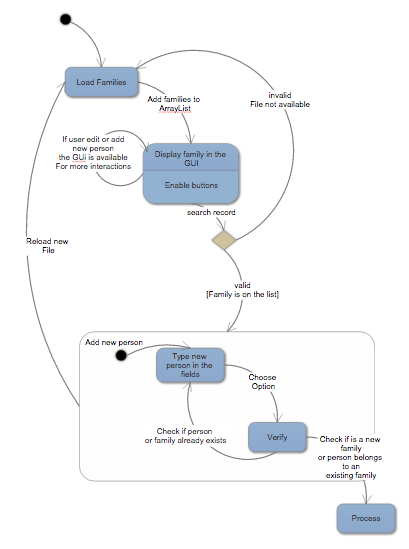
appendTextArea()

SaveToFile()

**UML**



**STD**



**Limitations:**

The program is meeting almost all requirements of the exercise, except that at the moment when adding a new user to an existing family, which would have a person with the same title from the one just added, it was suppose, to tell the user that it was not possible to do that, but at the moment when it happens it will append the new person several times in the file.

Due to time constraints I ended up delivering a file that I wish I could have improved better.

Among the changes:

\* I would modularize and split this file in different files.

\* Also, I would fix the bug that turn up when user try to add a person title that has already been added to the family.

\* If the user open the a file it would memorize the directory from the filepath

\* I really wanted to create validation for all fields in the user interface, but again no time

\* Finally I would use different classes, such as Person, Address and more instead of saving in a Hashmap in a completely different structure.

**Testing**

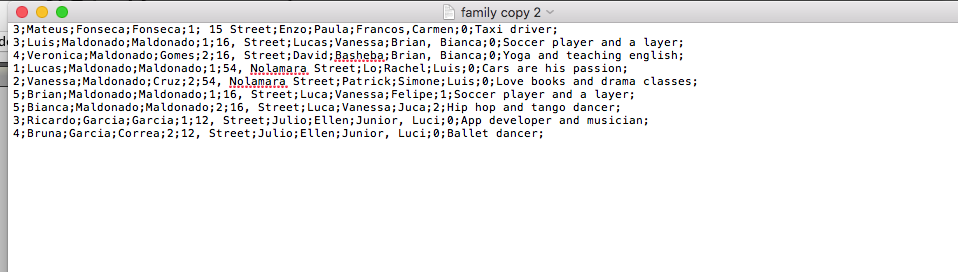
My strategy to test the app would be create test Classes that would implement and run all methods that I have created, on top of that I would really check on the .txt file to see if the modifications were stored permanently there.

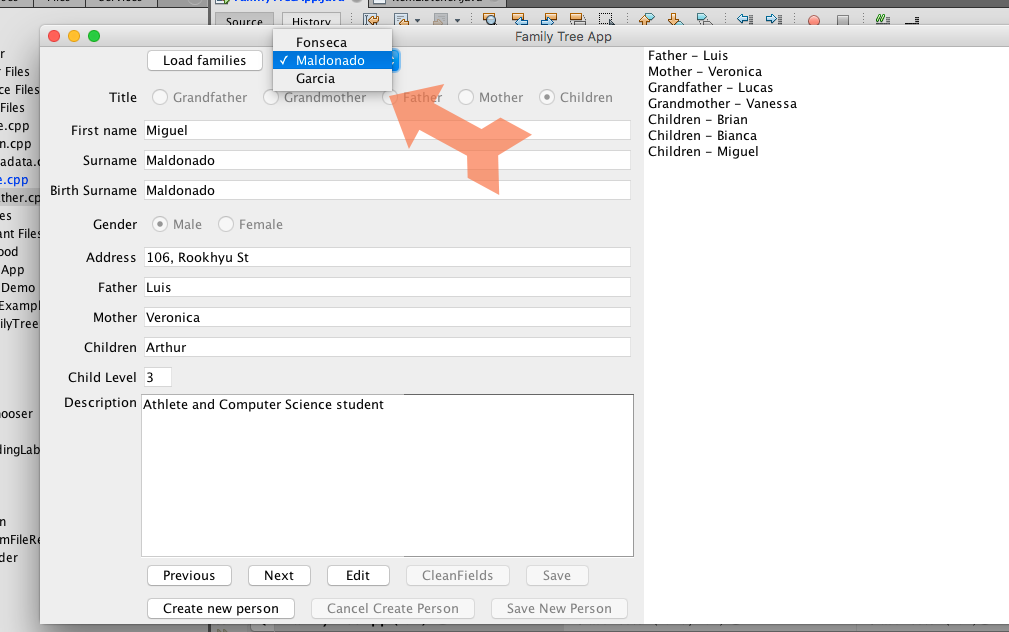
So despite the fact that I didn’t had the time to create all test classes, I test the app and it is working fine for all functions except if the user add a member to the same family and with the same family title, then there is a logical algorithm bug that I didn’t had enough time to fix it.

But for the working part, I will add the screen shots of the use of the program:

**Read from file**

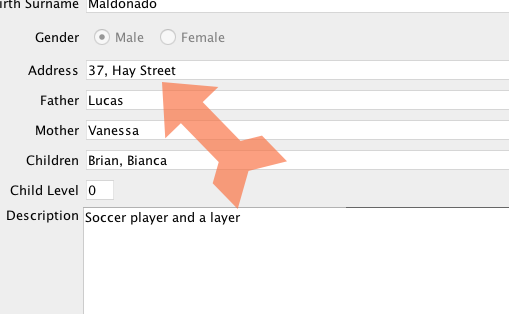
The following file has only 3 families when I load it would display all families and people fine.



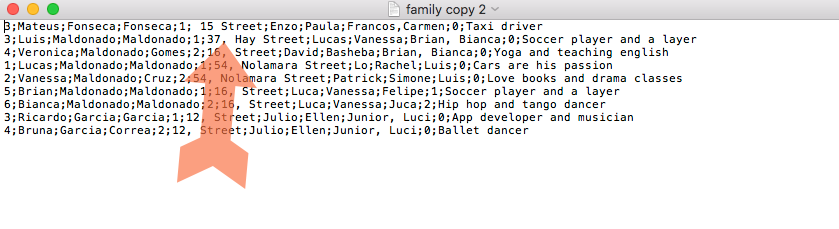


If I **edit** information it will also change in the file

For example in the next two pictures I have changed the address in the UI and it also changed in the file:

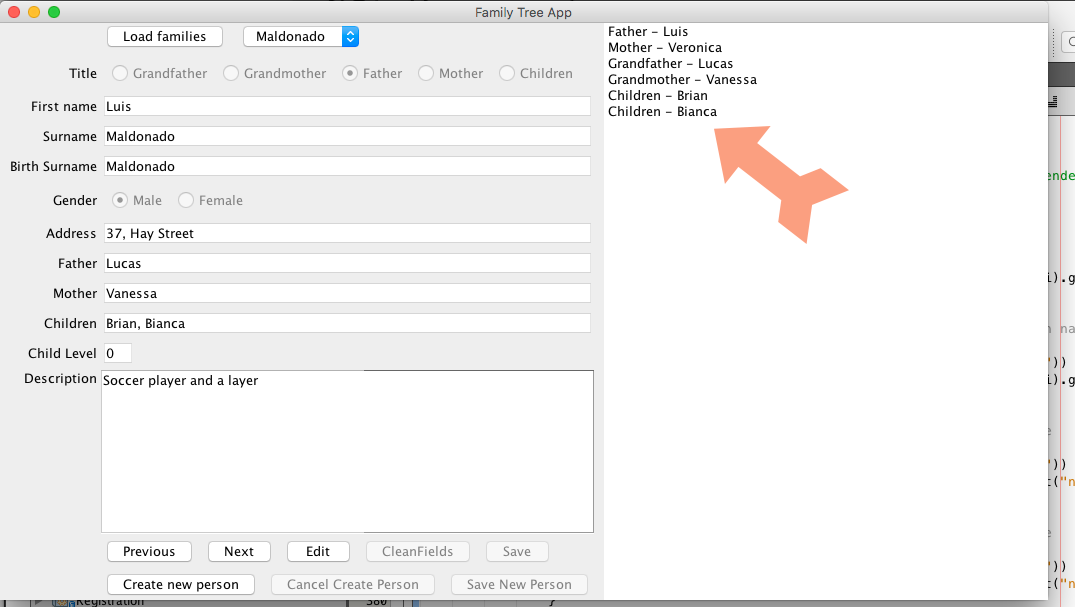


Below the file with the new address edit

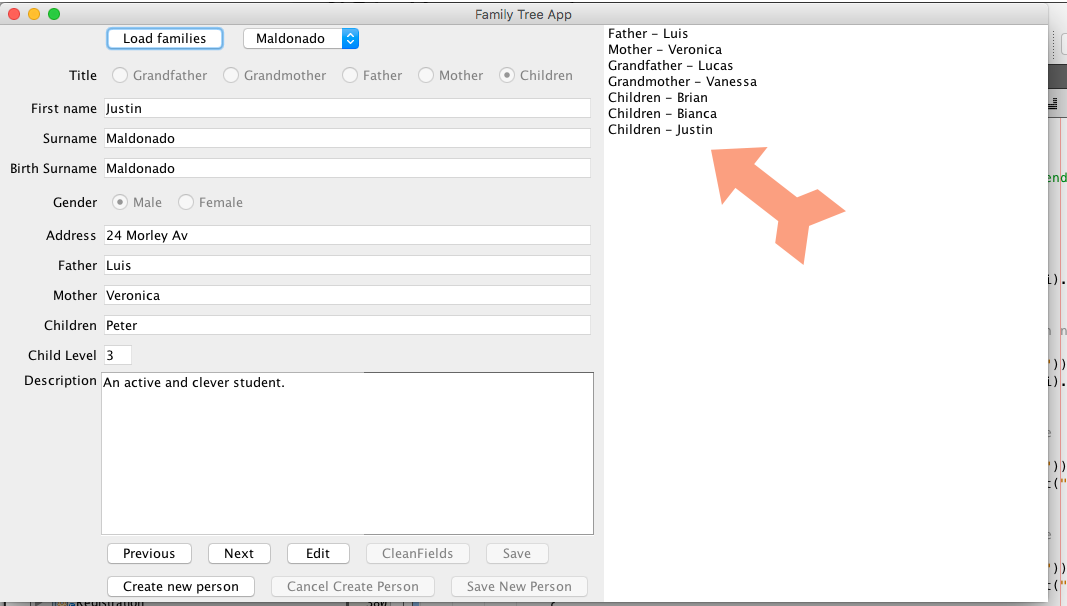


Also to **insert** new people it works fine, with the limitations that I have mentioned before

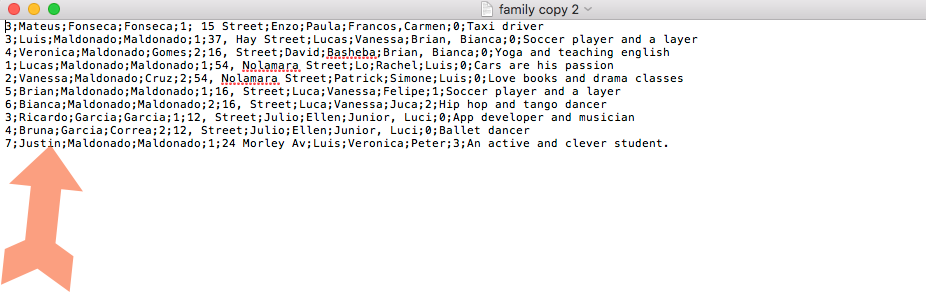
So this is the file before add a person to a family



After I insert a new children to the tree the person is displayed in the program



Also the file is **altered** with the new person included.



**Listings:**

/\*\*

\* Application that allow users to load a .txt file that have several

\* people in it and display a family tree of a particular selected family.

\* A Graphics user interface was created to easy management of family

\* member in the respective family tree. The program was written in Java. This

\* app allow users to :

\* <Title>

\* Family tree App

\* </title>

\* <ul>

\* <li>Load families from a file

\* <li>Display individuals in the interface as well as a tree from the family

\* <li>Allow edit people's information

\* <li>Add new families and people to the families.

\* <li>Travel from the root of the family to the last branch

\* <li>The current version and all previous versions of the program are available on a git repository

\* (see <a href="https://bitbucket.org/mikecamara/familytreeapp">Bit Bucket repository</a>)

\* </ul>

\* <p>

\* The program is meeting almost all requirements of the exercise, except that

\* at the moment when adding a new user to an existing family, which would have

\* a person with the same title from the one just added, it was suppose,

\* to tell the user that it was not possible to do that, but at the moment

\* when it happens it will append the new person several times in the file.

\* <p>

\* <ul>

\* Due to time constraints I ended up delivering a file that I wish I could have

\* improved better.

\* Among the changes:

\* <li>I would modularize and split this file in different files.

\* <li>Also, I would fix the bug that turn up when user try to add a person title

\* that has already been added to the family.

\* <li>If the user open the a file it would memorize the directory from the filepath

\* <li>Finnaly I would use different classes, such as Person, Address and more instead of saving in a Hashmap

\* in a completely different structure.

\* </ul>

\* %I% gets incremented each time you edit and delget a file

\* %G% is the date mm/dd/yy

\*

\* @author Mike Gomes

\* @version %I%, %G%

\* @since 1.0

\* File name: FamilyTreeApp.java

\* File to upload families: family.txt

\*/

package familytreeapp;

import java.awt.BorderLayout;

import java.awt.Dimension;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import java.util.Vector;

import javax.swing.ButtonGroup;

import javax.swing.DefaultComboBoxModel;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JRadioButton;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

import javax.swing.SwingUtilities;

import net.miginfocom.swing.MigLayout;

public class FamilyTreeApp {

/\*\*

\* Components of the user interface

\*/

private JFrame frame = new JFrame("Family Tree App");

private JPanel panel = new JPanel();

private JLabel lblFirstName = new JLabel("First name");

private JLabel lblSurname = new JLabel("Surname");

private JLabel lblTitle = new JLabel("Title");

private JLabel lblSurnameBirth = new JLabel("Birth Surname");

private JLabel lblGender = new JLabel("Gender");

private JLabel lblAddress = new JLabel("Address");

private JLabel lblFather = new JLabel("Father");

private JLabel lblMother = new JLabel("Mother");

private JLabel lblChildren = new JLabel("Children");

private JLabel lblChildrenLevel = new JLabel("Child Level");

private JLabel lblDsc = new JLabel("Description");

private JTextField txtfldName = new JTextField(20);

private JTextField txtfldSurname = new JTextField(20);

private JTextField txtfldBirthSurname = new JTextField(20);

private JTextField txtfldAddress = new JTextField(30);

private JTextField txtfldFather = new JTextField(20);

private JTextField txtfldMother = new JTextField(20);

private JTextField txtfldChildren = new JTextField(20);

private JTextField txtfldChildrenLevel = new JTextField(2);

private JButton btnPrevious = new JButton("Previous");

private JButton btnNext = new JButton("Next");

private JButton btnEdit = new JButton("Edit");

private JButton btnCreatePerson = new JButton("Create new person");

private JButton cleanFields = new JButton("CleanFields");

private JButton btnSave = new JButton("Save");

private JButton btnLoadFam = new JButton("Load families");

private JButton btnCancelAddPerson = new JButton("Cancel Create Person");

private JButton btnSaveSave = new JButton("Save New Person");

private final JRadioButton grandfatherJRadioButton; // selects plain text

private final JRadioButton grandmotherJRadioButton; // selects bold text

private final JRadioButton fatherJRadioButton; // selects italic text

private final JRadioButton motherJRadioButton; // bold and italic

private final JRadioButton childrenJRadioButton;

private final ButtonGroup radioGroup;

private final JRadioButton maleJRadioButton; // bold and italic

private final JRadioButton femaleJRadioButton;

private final ButtonGroup genderJRadioGroup;

private JComboBox<String> listOfFamiliesComboBox;

private Vector<String> myVector = new Vector<String>(1, 1);

private ArrayList<Map> familiesList = new ArrayList<Map>();

private Map<String, String> person;

private JFileChooser fc;

private String genderCheck, titleCheck;

private String family, familyNamePassed, familyAppended;

private String line;

private String filePath;

private BufferedReader in;

private JTextArea txaDsc = new JTextArea(10, 10);

private JTextArea txtAreaFamily = new JTextArea(33, 37);

private int counter,counterChildren, famNumber, token;

private int counterFamilyInner = 0;

private int counterComboBox = 0;

private int surnameCounter = 0;

private int newTitle = 0;

private int childLevelCheck = 0;

private int counterElemFamily = 0;

/\*\*

\* Choosing file method

\*/

private String selectFile() {

String dataFileText = null;

JFileChooser fc = new JFileChooser();

int returnVal = fc.showOpenDialog(null);

if (returnVal == JFileChooser.APPROVE\_OPTION) {

File file = fc.getSelectedFile();

if (file == null) {

return (dataFileText);

}

dataFileText = file.getAbsolutePath();

} else {

return (dataFileText);

}

return (dataFileText);

}// end select file method

/\*\*

\* This method simply disable edition of all UI fields

\*/

public void disableFields() {

fatherJRadioButton.setEnabled(false);

grandfatherJRadioButton.setEnabled(false);

grandmotherJRadioButton.setEnabled(false);

fatherJRadioButton.setEnabled(false);

motherJRadioButton.setEnabled(false);

childrenJRadioButton.setEnabled(false);

txtfldName.setEditable(false);

txtfldSurname.setEditable(false);

txtfldBirthSurname.setEditable(false);

maleJRadioButton.setEnabled(false);

femaleJRadioButton.setEnabled(false);

txtfldAddress.setEditable(false);

txaDsc.setEditable(false);

txtfldFather.setEditable(false);

txtfldMother.setEditable(false);

txtfldChildren.setEditable(false);

txtfldChildrenLevel.setEditable(false);

} // end disableFields method

/\*\*

\* This method clean fields and turn edition of fields enable

\*/

public void cleanFields() {

fatherJRadioButton.setEnabled(true);

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setEnabled(true);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setEnabled(true);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setEnabled(true);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

motherJRadioButton.setEnabled(true);

childrenJRadioButton.setEnabled(true);

childrenJRadioButton.setSelected(false);

txtfldName.setText("");

txtfldName.setEditable(true);

txtfldSurname.setText("");

txtfldSurname.setEditable(true);

txtfldBirthSurname.setText("");

txtfldBirthSurname.setEditable(true);

maleJRadioButton.setEnabled(true);

maleJRadioButton.setSelected(false);

femaleJRadioButton.setEnabled(true);

femaleJRadioButton.setSelected(false);

txtfldAddress.setEditable(true);

txtfldAddress.setText("");

txaDsc.setText("");

txaDsc.setEditable(true);

txtfldFather.setEditable(true);

txtfldFather.setText("");

txtfldMother.setEditable(true);

txtfldMother.setText("");

txtfldChildren.setEditable(true);

txtfldChildren.setText("");

txtfldChildrenLevel.setEditable(true);

txtfldChildrenLevel.setText("");

}// end cleanFields method

/\*\*

\* This methods return which radio button is selected out of 5 option

\* Grandfather, Grandmother, father, mother, children

\*/

public String getRadioButtonSelected(){

if (grandfatherJRadioButton.isSelected()) {

titleCheck = "1";

return "1";

} else if (grandmotherJRadioButton.isSelected()) {

titleCheck = "2";

return "2";

} else if (fatherJRadioButton.isSelected()) {

titleCheck = "3";

return "3";

} else if (motherJRadioButton.isSelected()) {

titleCheck = "4";

return "4";

} else {

titleCheck = "5";

return "5";

}

}// end get radio button choice method

/\*\*

\* this method get info given by the user in the UI fields

\* and create a map object person

\*/

public void addPerson() {

/\*\*

\* create a new HashMap object

\*/

person = new HashMap<String, String>();

/\*\*

\* insert fields of the person

\*/

person.put("title", titleCheck);

person.put("name", txtfldName.getText());

person.put("surname", txtfldSurname.getText());

person.put("birthSurname", txtfldBirthSurname.getText());

/\*\*

\* compares if male radio button is selected

\* if so, gender becomes male, if not gender female

\*/

if (maleJRadioButton.isSelected()) {

person.put("gender", "1");

} else {

person.put("gender", "2");

}

person.put("address", txtfldAddress.getText());

person.put("father", txtfldFather.getText());

person.put("mother", txtfldMother.getText());

person.put("children", txtfldChildren.getText());

person.put("childrenLevel", txtfldChildrenLevel.getText());

person.put("description", txaDsc.getText());

/\*\*

\* If title is 5 means it was a children

\*/

if (titleCheck.equals("5")) {

/\*\*

\* So I add the children Level to 4, it will keep family

\* ordered by date of Birth

\*/

newTitle = 4 + Integer.parseInt(txtfldChildrenLevel.getText());

/\*\*

\* add to the person the new value which is the sum of 4 with children level

\*/

person.put("title", String.valueOf(newTitle));

}

/\*\*

\* Finally add person to the ArrayList that contains all person

\*/

familiesList.add(person);

/\*\*

\* Disable button Save create new Person

\*/

btnSaveSave.setEnabled(false);

/\*\*

\* disable button save after edit

\*/

btnSave.setEnabled(false);

}// end add person method

/\*\*

\* Method to append people to their family in the text area in the right

\* hand side of the user interface

\*/

public void appendTextArea() {

/\*\*

\* Clean text area, so it still working when I change family

\*/

txtAreaFamily.setText("");

/\*\*

\* Get the current txtField value for surname

\*/

familyAppended = txtfldSurname.getText();

/\*\*

\* loop all person in the array

\*/

for (int i = 0; i < familiesList.size(); i++) {

/\*\*

\* If surname of the item iterator is same as textfield given

\*/

if (familiesList.get(i).get("surname").toString().equals(familyAppended)) {

/\*\*

\* If title == 1 adds word Grandfather to the left of the name

\*/

if (familiesList.get(i).get("title").toString().equals("1")) {

txtAreaFamily.append(" Grandfather - " + familiesList.get(i).get("name").toString() + "\n");

}

/\*\*

\* If title == 2 adds word grandmother to the left of the person name

\*/

else if (familiesList.get(i).get("title").toString().equals("2")) {

txtAreaFamily.append(" Grandmother - " + familiesList.get(i).get("name").toString() + "\n");

}

/\*\*

\* If title == 3 adds word father to the left of the person name

\*/

else if (familiesList.get(i).get("title").toString().equals("3")) {

txtAreaFamily.append(" Father - " + familiesList.get(i).get("name").toString() + "\n");

}

/\*\*

\* If title == 4 adds word mother to the left of the person name

\*/

else if (familiesList.get(i).get("title").toString().equals("4")) {

txtAreaFamily.append(" Mother - " + familiesList.get(i).get("name").toString() + "\n");

}

/\*\*

\* If title == 5 adds word children to the left of the person name

\*/

else {

txtAreaFamily.append(" Children - " + familiesList.get(i).get("name").toString() + "\n");

}

} // end comparing surname textField with iterator

}// end loop all person

} // end append text to the area family tree

/\*\*

\* method to save on file

\*/

public void SaveToFile() {

/\*\*

\* if there is a valid file path

\*/

if (filePath != null) {

try {

/\*\*

\* create a fileWriter object

\*/

FileWriter writer = new FileWriter(filePath);

/\*\*

\* loop all person in arrayList of person

\*/

for (int j = 0; j < familiesList.size(); j++) {

/\*\*

\* write all person in the arraylist in the file

\*/

writer.write(familiesList.get(j).get("title").toString());

writer.write(";");

writer.write(familiesList.get(j).get("name").toString());

writer.write(";");

writer.write(familiesList.get(j).get("surname").toString());

writer.write(";");

writer.write(familiesList.get(j).get("birthSurname").toString());

writer.write(";");

writer.write(familiesList.get(j).get("gender").toString());

writer.write(";");

writer.write(familiesList.get(j).get("address").toString());

writer.write(";");

writer.write(familiesList.get(j).get("father").toString());

writer.write(";");

writer.write(familiesList.get(j).get("mother").toString());

writer.write(";");

writer.write(familiesList.get(j).get("children").toString());

writer.write(";");

writer.write(familiesList.get(j).get("childrenLevel").toString());

writer.write(";");

writer.write(familiesList.get(j).get("description").toString());

writer.write("\n");

}

// flush and close file

// so next time it do it the file will be created from new

// not appended

writer.flush();

writer.close();

/\*\*

\* disable save button

\*/

btnSave.setEnabled(false);

} catch (IOException a) {

// If there is a problem writing the file get an error message

System.out.println(a.getMessage());

}

} else if (null != filePath) {

// something goes wrong with the file, tell the user

JOptionPane.showMessageDialog(null, "Cannot open file");

}

}// end save to file method

/\*\*

\* constructor of the user interface

\*/

public FamilyTreeApp() {

/\*\*

\* create a layout style MigLayout()

\* The most flexible and easy to use

\* Layout Managers

\*/

panel.setLayout(new MigLayout());

/\*\*

\* Create a comboBox menu that contains the surname off all families in the program

\*/

listOfFamiliesComboBox = new JComboBox<String>(myVector); // set up JComboBox

/\*\*

\* add a button to load files, skip let the button align better and

\* split2 open space for 2 buttons in the same line

\*/

panel.add(btnLoadFam, "skip, split2");

/\*\*

\* add the combobox menu in the UI, wrap skip to next line after that

\*/

panel.add(listOfFamiliesComboBox, "wrap , gap 10"); // add combo box to JFrame

/\*\*

\* add label with title, set position and spacing

\*/

panel.add(lblTitle, "align right, gap 10");

/\*\*

\* create 5 different radio buttons that will be used to select

\* title of the person in the family

\*/

grandfatherJRadioButton = new JRadioButton("Grandfather", true);

grandmotherJRadioButton = new JRadioButton("Grandmother", false);

fatherJRadioButton = new JRadioButton("Father", false);

motherJRadioButton = new JRadioButton("Mother", false);

childrenJRadioButton = new JRadioButton("Children", false);

/\*\*

\* create a new panel

\*/

JPanel inPanel = new JPanel();

/\*\*

\* add all radio buttons to the panel

\*/

inPanel.add(grandfatherJRadioButton);

inPanel.add(grandmotherJRadioButton);

inPanel.add(fatherJRadioButton);

inPanel.add(motherJRadioButton);

inPanel.add(childrenJRadioButton);

/\*\*

\* create logical relationship between JRadioButtons

\*/

radioGroup = new ButtonGroup(); // create ButtonGroup

radioGroup.add(grandfatherJRadioButton); // add grandfather to group

radioGroup.add(grandmotherJRadioButton); // add grandmother to group

radioGroup.add(fatherJRadioButton); // add father to group

radioGroup.add(motherJRadioButton); // add mother

radioGroup.add(childrenJRadioButton); // add children

/\*\*

\* add inner panel to the main panel and skip for next line

\*/

panel.add(inPanel, "wrap");

/\*\*

\* add all labels and text fields to the panel

\*/

panel.add(lblFirstName, "align right, gap 10");

panel.add(txtfldName, "wrap, pushx, growx");

panel.add(lblSurname, "align right, gap 10");

panel.add(txtfldSurname, "wrap, pushx, growx");

panel.add(lblSurnameBirth, "align right, gap 10");

panel.add(txtfldBirthSurname, "wrap, pushx, growx");

panel.add(lblGender, "align right, gap 10");

/\*\*

\* create two radio buttons to select person gender

\*/

maleJRadioButton = new JRadioButton("Male", true);

femaleJRadioButton = new JRadioButton("Female", false);

/\*\*

\* create inner panel to add both radio gender button

\*/

JPanel inPanelGender = new JPanel(); // Create new panel

/\*\*

\* Add components to inner pannel

\*/

inPanelGender.add(maleJRadioButton);

inPanelGender.add(femaleJRadioButton);

genderJRadioGroup = new ButtonGroup(); // create ButtonGroup

genderJRadioGroup.add(maleJRadioButton); // add male to group

genderJRadioGroup.add(femaleJRadioButton); // add female to group

panel.add(inPanelGender, "wrap");

/\*\*

\* add labels and text fields to the UI

\*/

panel.add(lblAddress, "align right, gap 10");

panel.add(txtfldAddress, "wrap, pushx, growx");

panel.add(lblFather, "align right, gap 10");

panel.add(txtfldFather, "wrap, pushx, growx");

panel.add(lblMother, "align right, gap 10");

panel.add(txtfldMother, "wrap, pushx, growx");

panel.add(lblChildren, "align right, gap 10");

panel.add(txtfldChildren, "wrap, pushx, growx");

panel.add(lblChildrenLevel, "align right, gap 10");

panel.add(txtfldChildrenLevel, "wrap");

/\*\*

\* add text area to user insert descrition of person

\*/

panel.add(lblDsc, "top, align right, gap 10");

/\*\*

\* make text area scrollable in case user type to much text

\*/

panel.add(new JScrollPane(txaDsc), "wrap, push, grow");

/\*\*

\* disable buttons before add them to panel

\*/

btnPrevious.setEnabled(false);

btnNext.setEnabled(false);

btnEdit.setEnabled(false);

/\*\*

\* Disable elements and add them all to panel

\*/

panel.add(btnPrevious, "skip, split5");

panel.add(btnNext);

panel.add(btnEdit);

cleanFields.setEnabled(false);

panel.add(cleanFields);

btnSave.setEnabled(false);

panel.add(btnSave, "wrap");

btnCreatePerson.setEnabled(false);

panel.add(btnCreatePerson, "skip, split3");

btnSaveSave.setEnabled(false);

btnCancelAddPerson.setEnabled(false);

panel.add(btnCancelAddPerson);

panel.add(btnSaveSave);

panel.add(txtAreaFamily, "east, gap 10");

/\*\*

\* add panel to the frame

\*/

frame.add(panel);

/\*\*

\* default close operation in case user close panel

\*/

frame.setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

frame.pack();

frame.setVisible(true);

/\*\*

\* Add actionlisteners for buttons section

\*

\* add actionListeners for when user press button Load Family

\*/

btnLoadFam.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* use method selectFile() to define a valid path

\*/

filePath = selectFile();

/\*\*

\* in case path is not invalid

\*/

if ((filePath != null)) {

try {

/\*\*

\* Create a BufferedReader object to read the filePath

\*/

BufferedReader in = new BufferedReader(new FileReader(filePath));

/\*\*

\* Create 3 temporary string tokens

\*/

String value, valueToken0, valueToken9;

/\*\*

\* read the file line by line

\*/

line = in.readLine();

/\*\*

\* loop through all lines of text file chosen (family.txt)

\*/

while (line != null) {

/\*\*

\* at each line create a new Hash map object which is a person

\*/

person = new HashMap<String, String>();

/\*\*

\* read first space and get title

\* from 1 to 5

\*/

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

valueToken0 = tokens[0];

person.put("title", valueToken0);

/\*\*

\* get name of the person

\*/

value = tokens[1];

person.put("name", value);

/\*\*

\* get surname

\*/

value = tokens[2];

person.put("surname", value);

/\*\*

\* get birthSurname

\*/

value = tokens[3];

person.put("birthSurname", value);

/\*\*

\* get Gender

\*/

value = tokens[4];

person.put("gender", value);

/\*\*

\* get person address

\*/

value = tokens[5];

person.put("address", value);

/\*\*

\* get person father

\*/

value = tokens[6];

person.put("father", value);

/\*\*

\* get person mother

\*/

value = tokens[7];

person.put("mother", value);

/\*\*

\* get person children

\*/

value = tokens[8];

person.put("children", value);

/\*\*

\* get person childrenLevel

\*/

valueToken9 = tokens[9];

person.put("childrenLevel", valueToken9);

/\*\*

\* get person description

\*/

value = tokens[10];

person.put("description", value);

/\*\*

\* check if is a children title == 5

\*/

if (valueToken0.equals("5")) {

/\*\*

\* if so adds ChildrenLevel(valueToken9)

\* to 4 so we can keep an ordered family

\* based on the birth order of the kids

\*/

newTitle = 4 + Integer.parseInt(valueToken9);

/\*\*

\* new person children gets new title

\*/

person.put("title", String.valueOf(newTitle));

}

/\*\*

\* adds the person read in the line to the

\* array of Person familiesList

\*/

familiesList.add(person);

/\*\*

\* jump to the next line

\*/

line = in.readLine();

}

in.close();// finish reading the file

}

/\*\*

\* handle file exceptions

\*/

catch (IOException a) {

System.out.println(a.getMessage());

}

} else if (null != filePath) {

}

/\*\*

\* Loops all maps (persons) stored in the arraylist

\*/

for (int i = 0; i < familiesList.size(); i++) {

/\*\*

\* I have a vector myVector that get the surname of the current iterate map (person)

\* The first comparison is if myVector already have the surname

\*/

if (myVector.contains(familiesList.get(i).get("surname").toString())) {

}

/\*\*

\* if it is not in the vector, in other words new

\* add the surname to the list of surnamens myVector

\*/

else {

myVector.add(familiesList.get(i).get("surname").toString());

}

} // end for loop adds surnames to vector from ComboBox menu

/\*\*

\* Update combobox menu

\*/

listOfFamiliesComboBox.setMaximumRowCount(myVector.size());

listOfFamiliesComboBox.setModel(new DefaultComboBoxModel(myVector));

/\*\*

\* families are loaded, so enable buttons to navigate left and right

\* of the family tree

\* and buttons to edit and create new person

\*/

btnPrevious.setEnabled(true);

btnNext.setEnabled(true);

btnEdit.setEnabled(true);

btnCreatePerson.setEnabled(true);

}// end action perfomed

} // end action listener button load family

);// end add/ action listener button load family

/\*\*

\* combobox button that contains all families available in the file

\*/

listOfFamiliesComboBox.addItemListener(new ItemListener() // anonymous inner class

{

/\*\*

\* handle JComboBox event

\*/

@Override

/\*\*

\* in case user flip among the families available

\*/

public void itemStateChanged(ItemEvent event) {

/\*\*

\* determine whether item selected

\*/

if (event.getStateChange() == ItemEvent.SELECTED) {

//add a counter

counterElemFamily = 0;

/\*\*

\* gets the family selected and pass the value to a string var

\*/

family = listOfFamiliesComboBox.getSelectedItem().toString();

/\*\*

\* loop all elements of the arrayList with People

\*/

for (int i = 0; i < familiesList.size(); i++) {

/\*\*

\* if surname of person iterator same as family chosen

\* in the menu

\*/

if (familiesList.get(i).get("surname").toString().equals(family)) {

/\*\*

\* Incremente counterElemFamily

\*/

counterElemFamily++;

/\*\*

\* check if title of the item iterator is the father

\* of the family title == 3

\*/

if (familiesList.get(i).get("title").toString().equals("3")) {

/\*\*

\* if it is the father then set the the radioButton

\* father to be enabled

\* and the other to be disabled

\*/

fatherJRadioButton.setSelected(true);

grandfatherJRadioButton.setEnabled(false);

grandmotherJRadioButton.setEnabled(false);

fatherJRadioButton.setEnabled(false);

motherJRadioButton.setEnabled(false);

childrenJRadioButton.setEnabled(false);

/\*\*

\* set all textfields with the father details

\* gathered from the iterator item

\*/

txtfldName.setText(familiesList.get(i).get("name").toString());

txtfldName.setEditable(false);

txtfldSurname.setText(familiesList.get(i).get("surname").toString());

txtfldSurname.setEditable(false);

txtfldBirthSurname.setText(familiesList.get(i).get("birthSurname").toString());

txtfldBirthSurname.setEditable(false);

if (familiesList.get(i).get("gender").toString().equals("1")) {

maleJRadioButton.setSelected(true);

} else {

femaleJRadioButton.setSelected(true);

}

maleJRadioButton.setEnabled(false);

femaleJRadioButton.setEnabled(false);

txtfldAddress.setText(familiesList.get(i).get("address").toString());

txtfldAddress.setEditable(false);

txaDsc.setText(familiesList.get(i).get("description").toString());

txaDsc.setEditable(false);

txtfldFather.setText(familiesList.get(i).get("father").toString());

txtfldFather.setEditable(false);

txtfldMother.setText(familiesList.get(i).get("mother").toString());

txtfldMother.setEditable(false);

txtfldChildren.setText(familiesList.get(i).get("children").toString());

txtfldChildren.setEditable(false);

txtfldChildrenLevel.setText(familiesList.get(i).get("childrenLevel").toString());

txtfldChildrenLevel.setEditable(false);

/\*\*

\* var counter get current number title

\* in this case will be always 3

\* the var counter will be used to jump

\* from one member to the other using

\* the previous and next buttons

\*/

counter = Integer.parseInt(familiesList.get(i).get("title").toString());

/\*\*

\* vara familyNamePassed will get the surname of the iterator

\*/

familyNamePassed = familiesList.get(i).get("surname").toString();

/\*\*

\* token gets the position of the person in the

\* ArrayList of persons

\*/

token = i;

} // end comparing title iterator == 3

}// end comparing surname to family taken from comboBox

}// end loop all elements of the ArrayList with person

/\*\*

\* This method print the family in the family tree text area

\*/

appendTextArea();

}// end selecting element from ComboBox

} // end anonymous inner class

}); // End add ActionListener to comboBox menu

/\*\*

\* Button previoud action Listener Method

\*/

btnPrevious.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* counter decrement jumping to the above level of family

\*/

counter--;

/\*\*

\* if counter is at least one

\*/

if (counter > 0) {

/\*\*

\* loop all person in the arrayList

\*/

for (int i = 0; i < familiesList.size(); i++) {

/\*\*

\* compare key surname of the object i to family name selected

\*/

if (familiesList.get(i).get("surname").toString().equals(familyNamePassed)) {

/\*\*

\* counter will represent the title of the person

\* example father, mother, etc

\*/

if (Integer.parseInt(familiesList.get(i).get("title").toString()) == counter) {

/\*\*

\* swich statement that select the proper radio button

\* accordind to the title of person in the iterator

\*/

switch (counter) {

/\*\*

\* select grandfather radiobutton

\*/

case 1:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(true);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(false);

;

break;

/\*\*

\* select grandmother button

\*/

case 2:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(true);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(false);

;

break;

/\*\*

\* select father radio button

\*/

case 3:

fatherJRadioButton.setSelected(true);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(false);

;

break;

/\*\*

\* select mother radio button

\*/

case 4:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(true);

childrenJRadioButton.setSelected(false);

;

break;

/\*\*

\* select children radiobutton

\*/

case 5:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(true);

;

break;

default:

;

break;

}// end switcher that select right radio button

/\*\*

\* set textfields of the UI according to person

\* in the iterator

\*/

txtfldName.setText(familiesList.get(i).get("name").toString());

txtfldName.setEditable(false);

txtfldSurname.setText(familiesList.get(i).get("surname").toString());

txtfldSurname.setEditable(false);

txtfldBirthSurname.setText(familiesList.get(i).get("birthSurname").toString());

txtfldBirthSurname.setEditable(false);

if (familiesList.get(i).get("gender").toString().equals("1")) {

maleJRadioButton.setSelected(true);

} else {

femaleJRadioButton.setSelected(true);

}

maleJRadioButton.setEnabled(false);

femaleJRadioButton.setEnabled(false);

txtfldAddress.setText(familiesList.get(i).get("address").toString());

txtfldAddress.setEditable(false);

txaDsc.setText(familiesList.get(i).get("description").toString());

txaDsc.setEditable(false);

txtfldFather.setText(familiesList.get(i).get("father").toString());

txtfldFather.setEditable(false);

txtfldMother.setText(familiesList.get(i).get("mother").toString());

txtfldMother.setEditable(false);

txtfldChildren.setText(familiesList.get(i).get("children").toString());

txtfldChildren.setEditable(false);

txtfldChildrenLevel.setText(familiesList.get(i).get("childrenLevel").toString());

txtfldChildrenLevel.setEditable(false);

token = i;

}// end comparing counter

}// end comparing surname

}// end for loop over all person

}// end if counter is at least one or greater

/\*\*

\* Check if counter == 1 which means is the grandfather

\* hence no more people to go previous

\*/

if (counter == 1) {

/\*\*

\* so previous button got disabled

\*/

btnPrevious.setEnabled(false);

}// end if counter == 0

/\*\*

\* if counter is lesser than the number of people in the family

\*/

if (counter < counterElemFamily) {

/\*\*

\* set button next enabled

\*/

btnNext.setEnabled(true);

}

}// end action perfomed

} // end action listener button previous

); // end button previous add action Listener method

/\*\*

\* button next member of the family action listener

\*/

btnNext.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* increment counter, so jumps to the next title

\*/

counter++;

/\*\*

\* loop all people in the arrayList of person

\*/

for (int i = 0; i < familiesList.size(); i++) {

/\*\*

\* compare key surname of the object i to family name selected

\*/

if (familiesList.get(i).get("surname").toString().equals(familyNamePassed)) {

/\*\*

\* compare title being iterated with counter number

\*/

if (Integer.parseInt(familiesList.get(i).get("title").toString()) == counter) {

/\*\*

\* check counter to display title accordingly

\*/

switch (counter) {

case 1:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(true);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(false);

;

break;

case 2:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(true);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(false);

;

break;

case 3:

fatherJRadioButton.setSelected(true);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(false);

;

break;

case 4:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(true);

childrenJRadioButton.setSelected(false);

;

break;

default:

fatherJRadioButton.setSelected(false);

grandfatherJRadioButton.setSelected(false);

grandmotherJRadioButton.setSelected(false);

fatherJRadioButton.setSelected(false);

motherJRadioButton.setSelected(false);

childrenJRadioButton.setSelected(true);

;

break;

}

/\*\*

\* set all fields from the user interface to the iterator item

\*/

txtfldName.setText(familiesList.get(i).get("name").toString());

txtfldName.setEditable(false);

txtfldSurname.setText(familiesList.get(i).get("surname").toString());

txtfldSurname.setEditable(false);

txtfldBirthSurname.setText(familiesList.get(i).get("birthSurname").toString());

txtfldBirthSurname.setEditable(false);

if (familiesList.get(i).get("gender").toString().equals("1")) {

maleJRadioButton.setSelected(true);

} else {

femaleJRadioButton.setSelected(true);

}

maleJRadioButton.setEnabled(false);

femaleJRadioButton.setEnabled(false);

txtfldAddress.setText(familiesList.get(i).get("address").toString());

txtfldAddress.setEditable(false);

txaDsc.setText(familiesList.get(i).get("description").toString());

txaDsc.setEditable(false);

txtfldFather.setText(familiesList.get(i).get("father").toString());

txtfldFather.setEditable(false);

txtfldMother.setText(familiesList.get(i).get("mother").toString());

txtfldMother.setEditable(false);

txtfldChildren.setText(familiesList.get(i).get("children").toString());

txtfldChildren.setEditable(false);

txtfldChildrenLevel.setText(familiesList.get(i).get("childrenLevel").toString());

txtfldChildrenLevel.setEditable(false);

/\*\*

\* counter becomes the one title from the item iterated

\*/

counter = Integer.parseInt(familiesList.get(i).get("title").toString());

/\*\*

\* tokes has the position of this person in the arraylist

\*/

token = i;

}// end of comparing title with counter

} // end of comparing surname

} //end for loop all person

/\*\*

\* if counter value == 2 button previous that could be disabled

\* becomes enabled

\*/

if (counter == 2) {

btnPrevious.setEnabled(true);

}

/\*\*

\* if counter is the same of members in the family

\* disable button next

\*/

if (counter == counterElemFamily) {

btnNext.setEnabled(false);

}

}// end action performed

}// end action listener methos to next button

); // end action listener method add

/\*\*

\* button edit action method

\*/

btnEdit.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* set all current UI fields to enable editing

\*/

fatherJRadioButton.setEnabled(true);

grandfatherJRadioButton.setEnabled(true);

grandmotherJRadioButton.setEnabled(true);

fatherJRadioButton.setEnabled(true);

motherJRadioButton.setEnabled(true);

childrenJRadioButton.setEnabled(true);

txtfldName.setEditable(true);

txtfldSurname.setEditable(true);

txtfldBirthSurname.setEditable(true);

maleJRadioButton.setEnabled(true);

femaleJRadioButton.setEnabled(true);

txtfldAddress.setEditable(true);

txaDsc.setEditable(true);

txtfldFather.setEditable(true);

txtfldMother.setEditable(true);

txtfldChildren.setEditable(true);

txtfldChildrenLevel.setEditable(true);

/\*\*

\* also enables button save and cleanfields

\*/

btnSave.setEnabled(true);

cleanFields.setEnabled(true);

}// end action performed

}); // end action listener button edit

/\*\*

\* button clean fields action listenr

\*/

cleanFields.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* this button only call one method to clean fields

\*/

cleanFields();

}

});// end clean fields action listner method add

/\*\*

\* button create person add action listener method

\*/

btnCreatePerson.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* clean fields

\*/

cleanFields();

/\*\*

\* enable button cancel add person

\*/

btnCancelAddPerson.setEnabled(true);

/\*\*

\* button save create new person

\*/

btnSaveSave.setEnabled(true);

}

});// end action listener add method to the button create new person

/\*\*

\* button cancel add person action listener

\*/

btnCancelAddPerson.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* Clean fields

\*/

cleanFields();

/\*\*

\* disable editing fields

\*/

disableFields();

/\*\*

\* disable cancel add new person button

\*/

btnCancelAddPerson.setEnabled(false);

/\*\*

\* button save create new person set disabled

\*/

btnSaveSave.setEnabled(false);

}

});

/\*\*

\* button save after editing action listener

\*/

btnSave.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* this following method, check if there was changes made

\* in the person displayed for edition

\* if so it replaces the changes in the arrayList

\* and them save back to a file

\*/

/\*\*

\* compare if Jradion button selected is the same title from the user in the arrayList

\*/

if (grandfatherJRadioButton.isSelected()) {

familiesList.get(token).replace("title", familiesList.get(token).get("title").toString(), "1");

}

if (grandmotherJRadioButton.isSelected()) {

familiesList.get(token).replace("title", familiesList.get(token).get("title").toString(), "2");

}

if (fatherJRadioButton.isSelected()) {

familiesList.get(token).replace("title", familiesList.get(token).get("title").toString(), "3");

}

if (motherJRadioButton.isSelected()) {

familiesList.get(token).replace("title", familiesList.get(token).get("title").toString(), "4");

}

if (childrenJRadioButton.isSelected()) {

familiesList.get(token).replace("title", familiesList.get(token).get("title").toString(), "5");

}

/\*\*

\* compare if info in fields given by user have changed for specific person, if so replace it.

\*/

if (familiesList.get(token).get("name").toString().equals(txtfldName.getText())) {

} else {

familiesList.get(token).replace("name", familiesList.get(token).get("name").toString(), txtfldName.getText());

}

if (familiesList.get(token).get("surname").toString().equals(txtfldSurname.getText())) {

} else {

familiesList.get(token).replace("surname", familiesList.get(token).get("surname").toString(), txtfldSurname.getText());

}

if (familiesList.get(token).get("birthSurname").toString().equals(txtfldBirthSurname.getText())) {

} else {

familiesList.get(token).replace("birthSurname", familiesList.get(token).get("birthSurname").toString(), txtfldBirthSurname.getText());

}

if (maleJRadioButton.isSelected()) {

familiesList.get(token).replace("gender", familiesList.get(token).get("gender").toString(), "1");

}

if (femaleJRadioButton.isSelected()) {

familiesList.get(token).replace("gender", familiesList.get(token).get("gender").toString(), "2");

}

if (familiesList.get(token).get("address").toString().equals(txtfldAddress.getText())) {

} else {

familiesList.get(token).replace("address", familiesList.get(token).get("address").toString(), txtfldAddress.getText());

}

if (familiesList.get(token).get("description").toString().equals(txaDsc.getText())) {

} else {

familiesList.get(token).replace("description", familiesList.get(token).get("description").toString(), txaDsc.getText());

}

if (familiesList.get(token).get("father").toString().equals(txtfldFather.getText())) {

} else {

familiesList.get(token).replace("father", familiesList.get(token).get("father").toString(), txtfldFather.getText());

}

if (familiesList.get(token).get("mother").toString().equals(txtfldMother.getText())) {

} else {

familiesList.get(token).replace("mother", familiesList.get(token).get("mother").toString(), txtfldMother.getText());

}

if (familiesList.get(token).get("children").toString().equals(txtfldChildren.getText())) {

} else {

familiesList.get(token).replace("children", familiesList.get(token).get("children").toString(), txtfldChildren.getText());

}

if (familiesList.get(token).get("childrenLevel").toString().equals(txtfldChildrenLevel.getText())) {

} else {

familiesList.get(token).replace("childrenLevel", familiesList.get(token).get("childrenLevel").toString(), txtfldChildrenLevel.getText());

}

/\*\*

\* after updating the arrayList with changes from editin

\* save changes back to a file

\*/

SaveToFile();

/\*\*

\* set button save from editing set to disable

\*/

btnSave.setEnabled(false);

/\*\*

\* clean fields button disable

\*/

cleanFields.setEnabled(false);

/\*\*

\* disable editing fields and append family to text area

\*/

disableFields();

appendTextArea();

}// end action performed

});// end button save after editing action listener method

/\*\*

\* Add Action Listener to the Button Save New Person

\*/

btnSaveSave.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

/\*\*

\* Loop all elements of the arrayList familiesList

\* Which means all person

\*/

for (int i = 0; i < familiesList.size(); i++) {

/\*\*

\* Check if the surname given in the textField

\* is the same as the current person in the iteration

\* Eventually it will try them all

\*/

if (familiesList.get(i).get("surname").toString().equals(txtfldSurname.getText())) {

/\*\*

\* This variable will store how many people have the same surname

\* I will use this var to check if would insert person

\* In an existing family or create a new one

\*/

surnameCounter++;

/\*\*

\* Check if grandfather is the option checked in the Radio buttons

\*/

if (grandfatherJRadioButton.isSelected()) {

/\*\*

\* If they are the same I get the title and compare to 1

\* The Logic is the user is trying to create a

\* person that is already exists

\* Like a new father, where we would have one already

\*/

if (familiesList.get(i).get("title").toString().equals("1")) {

/\*\*

\* So if it the same alert the user

\*/

JOptionPane.showMessageDialog(null, "First delete the previous grandfather");

} else {

/\*\*

\* In this else statement

\* if it falls here it means that the title given

\* is different from 1

\* titleCheck variable is initiated

\*/

titleCheck = "1";

/\*\*

\* Add Person to arrayList of person

\*/

addPerson();

/\*\*

\* Save the new person to the file

\*/

SaveToFile();

/\*\*

\* Append the family in the text area

\*/

appendTextArea();

}// end esle

// }// end comparing surname

}// end grandfather radio button is selected

/\*\*

\* if grandfather button is not selected check if

\* grandmother radio button is selected

\*/

else if (grandmotherJRadioButton.isSelected()) {

/\*\*

\* If they are the same I get the title and compare to 2

\* The Logic is the user is trying to create a

\* person that is already exists

\* Like a new father, where we would have one already

\*/

if (familiesList.get(i).get("title").toString().equals("2")) {

// So if it the same alert the user

JOptionPane.showMessageDialog(null, "First delete the previous grandmother");

} else {

/\*\*

\* In this else statement

\* if it falls here it means that the title given

\* is different from 1

\* titleCheck variable is

\*/

titleCheck = "2";

/\*\*

\* Add Person to arrayList of person

\*/

addPerson();

/\*\*

\* Save the new person to the file

\*/

SaveToFile();

/\*\*

\* Append the family in the text area

\*/

appendTextArea();

}// End else

// }// End comparing surname

} // end grandmother is selected

/\*\*

\* if grandmother button is not selected check if

\* father radio button is selected

\*/

else if (fatherJRadioButton.isSelected()) {

/\*\*

\* If they are the same I get the title and compare to 3

\* The Logic is the user is trying to create a

\* person that is already exists

\* Like a new father, where we would have one already

\*/

if (familiesList.get(i).get("title").toString().equals("3")) {

// So if it the same alert the user

JOptionPane.showMessageDialog(null, "First delete the previous father");

} else {

/\*\*

\* In this else statement

\* if it falls here it means that the title given

\* is different from 1

\* titleCheck variable is initiated

\*/

titleCheck = "3";

/\*\*

\* Add Person to arrayList of person

\*/

addPerson();

/\*\*

\* Save the new person to the file

\*/

SaveToFile();

/\*\*

\* Append the family in the text area

\*/

appendTextArea();

}// end else

// }// end getting surname

}// end Father JRadio button selected

/\*\*

\* if father button is not selected check if

\* mother radio button is selected

\*/

else if (motherJRadioButton.isSelected()) {

/\*\*

\* If they are the same I get the title and compare to 4

\* The Logic is the user is trying to create a

\* person that is already exists

\* Like a new father, where we would have one already

\*/

if (familiesList.get(i).get("title").toString().equals("4")) {

// So if it the same alert the user

JOptionPane.showMessageDialog(null, "First delete the previous mother");

} else {

/\*\*

\* In this else statement

\* if it falls here it means that the title given

\* is different from 1

\* titleCheck variable is initiated

\*/

titleCheck = "4";

/\*\*

\* Add Person to arrayList of person

\*/

addPerson();

/\*\*

\* Save the new person to the file

\*/

SaveToFile();

/\*\*

\* Append the family in the text area

\*/

appendTextArea();

} // End else

// }// end comparing surname

}// end mother button is selected

/\*\*

\* if mother button is not selected check if

\* children radio button is selected

\*/

else if (childrenJRadioButton.isSelected()) {

/\*\*

\* If they are the same I get the title and compare to 5

\* The Logic is the user is trying to create a

\* person that is already exists

\* Like a new father, where we would have one already

\*/

if (familiesList.get(i).get("title").toString().equals("5")) {

/\*\*

\* Count how many children does the family have

\*/

counterChildren++;

/\*\*

\* check if counter lesser than one which

\* would mean that there is no other children

\* in the family

\*/

if (counterChildren < 1) {

/\*\*

\* So add the new children

\*/

addPerson();

/\*\*

\* Append the family in the text area

\*/

appendTextArea();

}

/\*\*

\* if there is at least one or more children in

\* the family it compares the children level

\* given by the user in the textfield

\* with the person in the iterator

\*/

else if (familiesList.get(i).get("childrenLevel").toString().equals(txtfldChildrenLevel.getText())) {

/\*\*

\* It would mean that there is already one children

\* in the same level as the one give

\* So if it alerts the user

\*/

JOptionPane.showMessageDialog(null, "First delete the previous children with same level");

} else {

/\*\*

\* In this else statement

\* if it falls here it means that the title given

\* is 5 and there is no othe children

\* with the same children Level

\* titleCheck variable is initiated

\*/

titleCheck = "5";

/\*\*

\* Add Person to arrayList of person

\*/

addPerson();

/\*\*

\* save to a file

\*/

SaveToFile();

/\*\*

\* apend family to the text area

\*/

appendTextArea();

}// end else same childrenLevel

} // end if title given == 5

// } // end comparing surname

} //end children button radio is checked

}// end checking surname

} // end loop for over all person in the arrayLis

/\*\*

\* surname is a counter is zero as the iterator loop all

\* person and didn't find the same surname

\*/

if (surnameCounter == 0) {

/\*\*

\* so in this case let's initialize a new person

\* in a new family

\* the following if else statement

\* gets which title does a person hav

\*/

getRadioButtonSelected();

/\*\*

\* Add Person to arrayList of person

\*/

addPerson();

/\*\*

\* as it is a new surname add it to the myVector

\* which holds all surnames in the comboBox

\*/

myVector.add(txtfldSurname.getText());

/\*\*

\* update size of combobox

\*/

listOfFamiliesComboBox.setMaximumRowCount(myVector.size());

/\*\*

\* updade and add new surname to the comboBox dropdown menu

\*/

listOfFamiliesComboBox.setModel(new DefaultComboBoxModel(myVector));

/\*\*

\* save to the file

\*/

SaveToFile();

/\*\*

\* append new person to the text area

\*/

appendTextArea();

} // end surnamecounter == 0

/\*\*

\* disable button save create new person

\*/

btnSaveSave.setEnabled(false);

/\*\*

\* disable button cancel creating a new person

\*/

btnCancelAddPerson.setEnabled(false);

/\*\*

\* disable edition of all fields

\*/

disableFields();

} // end action performed

} // end action listener

); // end add(actionlistener)

}// end FamilyTreeApp()

/\*\*

\* Client interface, this runs in the client computer

\*/

public static void main(String[] args) {

/\*\*

\* this method was required for the layout manager that I've chosen

\*/

SwingUtilities.invokeLater(new Runnable() {

/\*\*

\* override run method, required by layout manager used

\*/

@Override

public void run() {

new FamilyTreeApp();

}

});

}// end main method

} // end familyTreeApp class

// JCRJ