|  |  |
| --- | --- |
| **ELECTRONIC ASSIGNMENT COVERSHEET** |  |

|  |  |
| --- | --- |
| Student Number | 33679278 |
| Surname | Vu Hoang |
| Given name | Quan |
| Email | vuhoangquan1999@gmail.com |
|  |  |
| Unit Code | ICT373 |
| Unit name | Software Architectures |
| Enrolment mode |  |
| Date | 16/11/2020 |
| Assignment number | 2 |
| Assignment name | Assignment 2 |
| Tutor | Siew Cheong Chong |

|  |
| --- |
| Student’s Declaration:   * Except where indicated, the work I am submitting in this assignment is my own work and has not been submitted for assessment in another unit. * This submission complies with Murdoch University's academic integrity commitments. I am aware that information about plagiarism and associated penalties can be found at http://www.murdoch.edu.au/teach/plagiarism/. If I have any doubts or queries about this, I am further aware that I can contact my Unit Coordinator prior to submitting the assignment. * I acknowledge that the assessor of this assignment may, for the purpose of assessing this assignment:   + reproduce this assignment and provide a copy to another academic staff member; and/or   + submit a copy of this assignment to a plagiarism-checking service. This web-based service may retain a copy of this work for the sole purpose of subsequent plagiarism checking, but has a legal agreement with the University that it will not share or reproduce it in any form. * I have retained a copy of this assignment. * I will retain a copy of the notification of receipt of this assignment. If you have not received a receipt within three days, please check with your Unit Coordinator. |
| I am aware that I am making this declaration by submitting this document electronically and by using my Murdoch ID and password it is deemed equivalent to executing this declaration with my written signature. | |
| Optional Comments to Tutor:  E.g. If this is a group assignment, list group members here | |

*If you can, please insert this completed form into the body of* ***each*** *assignment you submit. Follow the instructions in the Unit Information and Learning Guide about how to submit your file(s) and how to name them, so the Unit Coordinator knows whose work it is.*

ICT373 Software Architectures

TSA 2020 - Assignment 2

Vu Hoang Quan – 33679278

Contents

[Specification 2](#_Toc56467093)

[User guide 3](#_Toc56467094)

[Structure and design 3](#_Toc56467095)

[Class diagram (open with StarUML): 3](#_Toc56467096)

[Function calls: 3](#_Toc56467097)

[Assumptions: 4](#_Toc56467098)

[Limitation 4](#_Toc56467099)

[Testing 4](#_Toc56467100)

[Source code listing 7](#_Toc56467101)

[public class As2ict373 extends Application { 8](#_Toc56467102)

[public class personHandler { 18](#_Toc56467103)

[public class person implements Serializable { 23](#_Toc56467104)

# Specification

This program shows a family tree view of all member and their relationship to each other. It has a proper GUI based on JavaFX 8 version. The program has 2 main tabs:

* A view tab where overall family tree can be viewed, and each person specific information will be shown when clicked on that person name.
* An edit tab where a new person can be added to the family tree.

The program will show an empty tree when started and can add a sample data with a quick button for fast viewing and testing. If the tree is empty, from edit tab, the new person will be added as root, otherwise the new person must have relation to existing person. The tree can export and import data from file (in ArrayList<person> type, save in a .dat file). Each file contain data for 1 family tree. If multiple family tree needs to be saved, then the files must be created manually before it can be saved by the program. The program can read or write data to a chosen file at any location.

# User guide

The program can be run from “As2ict373.java” file which contain the main. The program was created with NetBeans using JavaFX therefore is should work by import these file to a new project, then select “run file” inside “As2ict373.java” by right-click. Data file for save or read to the program must be in “.dat” format. The “.dat” data file should not be manually modified, or it may corrupt the data. “.dat” does not need to be added into project.

# Structure and design

## Class diagram (open with StarUML):

familyTreeClassDesign.mdj

This program has 2 classes and a main. the personHandler class taking care of UI and I/O to the person class. Main() will call instance of personHandler class. Main class (As2ict373.java) handle design and arrangement of the tabs and buttons location as well as action when click on these button or label. Person class contain basic information of 1 person and pointers as relation to other people (1 pointer for spouse, 1 for father, 1 for mother and an Array List for children(s). When a new person is added, every immediate relative to this person will change their relation accordingly.

## Function calls:

* As2ict373.java
  + start(Stage topView)
    - createUI(topView)
      * createPersonTree(new ArrayList<>())
        + showPersonInfo(event, treeView)
      * createViewUI(topView)
        + clickLoadSample()

getSample()

* + - * + clickSaveToFile(topView)

fileChooserDialogWindow(topView, message)

* + - * + clickReadFromFile(topView)

fileChooserDialogWindow(topView, message)

reloadTreeView()

* + - * createEditUI()
        + doClick(evt)

reloadTreeView

* personHandler.java
  + generateFamilyTree()
    - familyTreeUI()
      * has1ChildAndIsCallingPerson()
* person.java

createPersonTree function will return a nested TreeItem type for ViewTab.

showPersonInfo() run when clicked on a person in the TreeView (can click on text or spaces next to it)

createViewUI() show specific information of 1 person and 3 button to get a sample as data, save or write family tree’s data to file. These buttons use FileChoose to choose specific file to read/write to.

createEditUI() show a form to add new people to the family tree. doClick() runs when clicked on button “Add this new person to Tree”.

## Assumptions:

* person’s address is simplified to just a simple String
* instead of showing up 1 level in the tree, the whole tree is shown by recursive call
* edit tab should add new people into tree not changing existing people (or removing)
* add new person from empty tree by let "relationship with" and "relationship type" empty
* a new person must have at least firstname, gender
* a couple is 1 male and 1 female (male spouse will be female and vice versa)
* each person can only have 0 or 1 spouse
* spouse person will have different parent (did not enforce this)
* if a person already has child(s) and that person has a spouse, both will have the same child(s)
* each person will only be shown in the tree once
  + if a child has 2 parents then the parents will not show each other as spouse
  + if a parent has 1 child and that child has already shown in the tree, then the parent will not be showing this child again (vice versa)
* a person can have same firstname, surname, gender but is considered to be different person (their relationship could be different)

# Limitation

* in Edit tab, if the family is not empty, the added person relation is empty, it will still be added to program but will not shows up in the tree view.
* in Edit tab, if the family is not empty, the “relationship with” person name is not found, the new person will still be added to program but will not shows up in the tree view.
* Cannot delete existing people or changing them from GUI.
* In Tree View, labels such as “parent:” will show as empty person with bio as “parent:”

# Testing

### Test table:

|  |  |  |
| --- | --- | --- |
| **Test description** | **Actual result** | **Passed/failed?** |
| Load empty tree when program start | Tree show as “empty tree” | Yes |
| load person info when clicked on person name in Tree View | Empty info field show “unspecified” as default | Yes |
| Edit: add person as root from empty tree | Worked. “relationship with” and “relationship type” field must be left blank | Yes |
| Edit: add person based on relation from non-empty tree | new person added to data but will not affect Tree view | Yes |
| Edit: add person based on relation from empty tree | This person will be added to data but will not affect Tree view | Yes |
| Load sample tree, tree is empty | Sample data loaded | Yes |
| Load sample tree, tree is not empty | Sample data loaded but current family data lost | Yes |
| Save tree’s data to file | Open a window to choose file and save to that file | Yes |
| Read tree’s data from file | Open a window to choose file and read from that file, then reload tree views | Yes |
| Start on a new family tree | Close program and re-run it | Yes |

Sample tests:

The GUI when start the program:

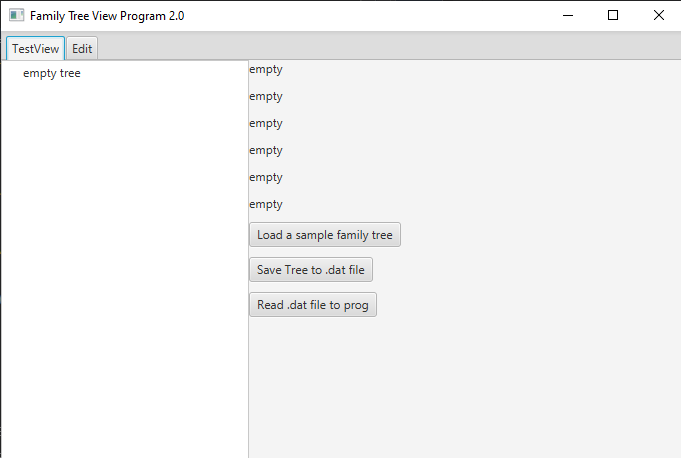
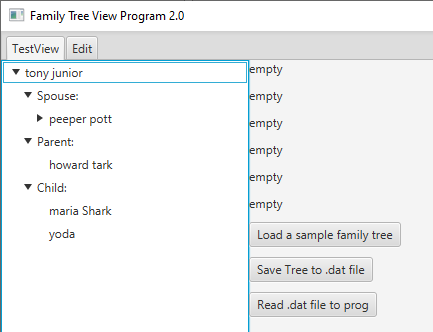
 

Figure 1 when start Figure 2 clicked on "load a sample.."

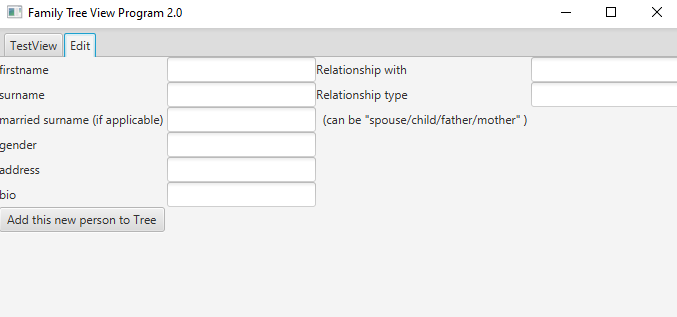
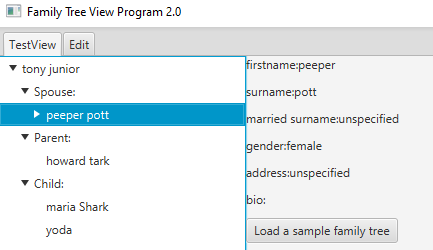
 

Figure 3 edit tab Figure 4 clicked on a person

Click on “Save Tree to .dat file” or “Read .dat from file to prog” will open a window to choose the file, default location of this windows is “. \\src\\data”

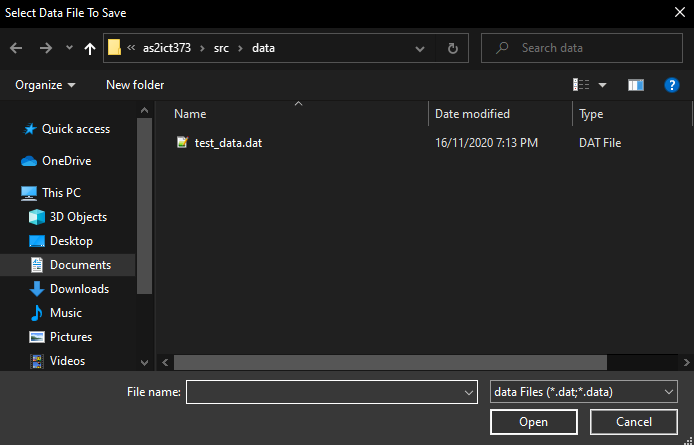


Figure 5 clicked on "save Tree..."

Add a person to empty tree:

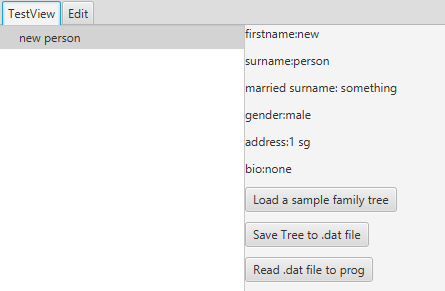
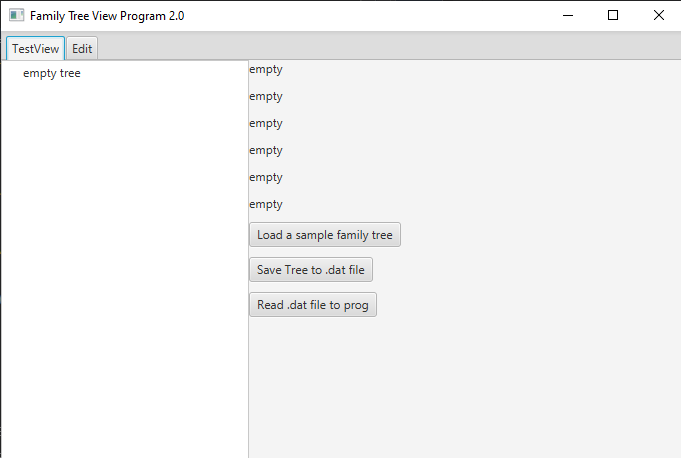


Figure 6 before add Figure 6.1 after add

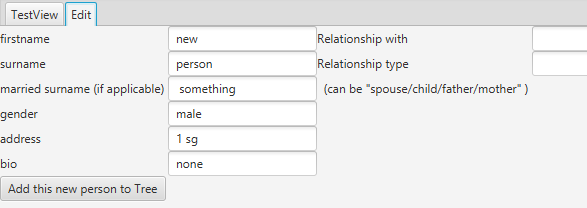
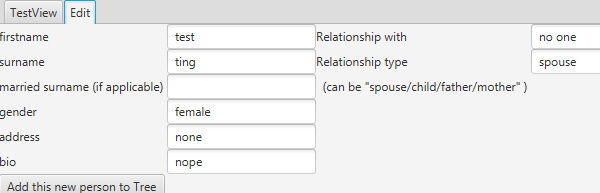


Figure 7 in edit tab

If add a person but “relationship with” is a name that not exist.



Printed to output:



That person will not be added to tree.

# Source code listing

package as2ict373;

import javafx.application.Application;

import javafx.stage.Stage;

import javafx.scene.\*;

import javafx.scene.control.\*;

import javafx.scene.layout.\*;

import javafx.event.\*;

import javafx.geometry.Orientation;

import javafx.scene.input.MouseEvent;

import javafx.scene.text.Text;

import javafx.stage.FileChooser;

import javafx.stage.FileChooser.ExtensionFilter;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.util.ArrayList;

/\*\*

\*

\* @author Vu Hoang Quan

\*/

### public class As2ict373 extends Application {

private personHandler personInterface = new personHandler();

private FlowPane fPane = new FlowPane();

private Label UI\_person\_firstname\_label, UI\_person\_surname\_label,

UI\_person\_married\_surname\_label, UI\_person\_address\_label,

UI\_person\_bio\_label, UI\_person\_gender\_label;

private TextField UI\_edit\_person\_firstname, UI\_edit\_person\_surname,

UI\_edit\_person\_gender, UI\_edit\_person\_address,

UI\_edit\_person\_married\_surname, UI\_edit\_person\_bio,

UI\_edit\_person\_relation, UI\_edit\_existed\_person;

private Button Submit\_new\_person, Export\_data\_to\_file, Import\_data\_from\_file,

Load\_sample\_data;

@Override

public void start(Stage topView) {

createUI(topView);

topView.show();

}

public static void main(String[] argv) {

launch(argv);

}

// design of 2 tab, 2 collumns on 1st tab

private void createUI(Stage topView) {

TabPane tp = new TabPane();

tp.tabClosingPolicyProperty();

fPane.getChildren().add(createPersonTree(new ArrayList<>()));

fPane.getChildren().add(createViewUI(topView));

Tab tabView = new Tab("TestView", fPane);

Tab tabEdit = new Tab("Edit", createEditUI());

tabView.setClosable(false);

tabEdit.setClosable(false);

tp.getTabs().addAll(tabView, tabEdit);

topView.setScene(new Scene(new FlowPane(Orientation.HORIZONTAL)));

topView.setTitle("Family Tree View Program 2.0");

((FlowPane) (topView.getScene().getRoot())).getChildren().add(tp);

}

// left collumn of view tab's design

private TreeView<person> createPersonTree(ArrayList<person> data) {

//handle empty Tree

if (data.isEmpty()) {

TreeView<person> treeView = new TreeView<>(new TreeItem<>(new person("empty tree")));

return treeView;

} else {

personInterface.setData(data);

person searchPerson = personInterface.getData().get(0);//1st person added = root

//searchPerson = personInterface.getPersonWithName("may");

//hard code this function to test tree view with different person as tree root

TreeView<person> treeView = new TreeView<>(personInterface.generateFamilyTree(searchPerson));

EventHandler<MouseEvent> mouseEventHandle = (MouseEvent event) -> {

showPersonInfo(event, treeView);

};

//create tree view to contain the root

treeView.addEventHandler(MouseEvent.MOUSE\_CLICKED, mouseEventHandle);

return treeView;

}

}

// action to perform when clicked on family tree

private void showPersonInfo(MouseEvent event, TreeView<person> treeView) {

Node node = event.getPickResult().getIntersectedNode();

// Accept clicks only on node cells, and not on empty spaces of the TreeView

if ((node instanceof Text || node instanceof TreeCell && ((TreeCell) node).getText() != null)) {

String name = (String) ((TreeItem) treeView.getSelectionModel().getSelectedItem()).toString();

person name2 = (person) ((TreeItem) treeView.getSelectionModel().getSelectedItem()).getValue();

UI\_person\_firstname\_label.setText("firstname:" + name2.getFirstname());

UI\_person\_surname\_label.setText("surname:" + name2.getSurname());

UI\_person\_married\_surname\_label.setText("married surname:" + name2.getMarriedSurname());

UI\_person\_gender\_label.setText("gender:" + name2.getGender());

UI\_person\_address\_label.setText("address:" + name2.getAddress());

UI\_person\_bio\_label.setText("bio:" + name2.getBio());

}

}

// right collumn of view tab's design

private Parent createViewUI(Stage topView) {

VBox layoutMgr = new VBox(10);

UI\_person\_firstname\_label = new Label("empty");

UI\_person\_surname\_label = new Label("empty");

UI\_person\_gender\_label = new Label("empty");

UI\_person\_married\_surname\_label = new Label("empty");

UI\_person\_address\_label = new Label("empty");

UI\_person\_bio\_label = new Label("empty");

Export\_data\_to\_file = new Button("Save Tree to .dat file");

Import\_data\_from\_file = new Button("Read .dat file to prog");

Load\_sample\_data = new Button("Load a sample family tree");

layoutMgr.getChildren().addAll(

UI\_person\_firstname\_label,

UI\_person\_surname\_label,

UI\_person\_married\_surname\_label,

UI\_person\_gender\_label,

UI\_person\_address\_label,

UI\_person\_bio\_label,

Load\_sample\_data,

Export\_data\_to\_file,

Import\_data\_from\_file);

Load\_sample\_data.setOnMouseClicked(value -> clickLoadSample());

Export\_data\_to\_file.setOnMouseClicked(value -> clickSaveToFile(topView));

Import\_data\_from\_file.setOnMouseClicked(value -> clickReadFromFile(topView));

return layoutMgr;

}

//action when click "Load a sample family tree"

private void clickLoadSample() {

//will change from createPersonTree(new TreeView<person>)

// to createPersonTree(getSample())

//reload Tree View with Sample

fPane.getChildren().remove(0);

fPane.getChildren().add(0, createPersonTree(getSample()));

}

//action when click "Save Tree to .dat file"

private void clickSaveToFile(Stage topView) {

try (ObjectOutputStream outputfile = new ObjectOutputStream(new FileOutputStream(

fileChooserDialogWindow(topView, "Select Data File To Save")))) {

outputfile.writeObject(personInterface.getData());

// save entire ArrayList from personHandler to outputfile

System.out.println("end of save data");

} catch (Exception ex) {

System.out.println("failed to save data " + ex);

}

}

//action when click "Read .dat file to prog"

private void clickReadFromFile(Stage topView) {

try (ObjectInputStream infile = new ObjectInputStream(new FileInputStream(

fileChooserDialogWindow(topView, "Select Data File To Read")))) {

personInterface.setData((ArrayList<person>) infile.readObject());

reloadTreeView();

System.out.println("end of read data");

} catch (Exception ex) {

System.out.println("failed to read data " + ex);

}

}

//reload Tree View

private void reloadTreeView() {

fPane.getChildren().remove(0);

fPane.getChildren().add(0, createPersonTree(personInterface.getData()));

}

//popup window to choose file

private String fileChooserDialogWindow(Stage topView, String message) {

FileChooser fileChooserWindow = new FileChooser();

fileChooserWindow.setTitle(message);

fileChooserWindow.getExtensionFilters().addAll(

new ExtensionFilter("data Files", "\*.dat", "\*.data"),

new ExtensionFilter("Text Files", "\*.txt"),

new ExtensionFilter("All Files", "\*.\*"));

//this is default fileChooser location which maybe different

File defaultFolder = new File(".\\src\\data");

fileChooserWindow.setInitialDirectory(defaultFolder);

File selectedFile = fileChooserWindow.showOpenDialog(topView);

if (selectedFile != null) {

//topView.display(selectedFile);

System.out.println(selectedFile.toString());

return selectedFile.getAbsolutePath();

} else {

return "";

}

}

//edit tab's design

private Node createEditUI() {

GridPane gPane = new GridPane();

UI\_edit\_person\_firstname = new TextField();

UI\_edit\_person\_surname = new TextField();

UI\_edit\_person\_gender = new TextField();

UI\_edit\_person\_address = new TextField();

UI\_edit\_person\_married\_surname = new TextField();

UI\_edit\_person\_bio = new TextField();

UI\_edit\_existed\_person = new TextField();

UI\_edit\_person\_relation = new TextField();

Submit\_new\_person = new Button("Add this new person to Tree");

try {

Submit\_new\_person.setOnAction(value -> doClick(value));

} catch (Exception ex) {

System.out.println("invalid input of new person to be added to family tree");

}

gPane.addRow(0, new Label("firstname "), UI\_edit\_person\_firstname);

gPane.addRow(1, new Label("surname "), UI\_edit\_person\_surname);

gPane.addRow(2, new Label("married surname (if applicable) "), UI\_edit\_person\_married\_surname);

gPane.addRow(3, new Label("gender "), UI\_edit\_person\_gender);

gPane.addRow(4, new Label("address "), UI\_edit\_person\_address);

gPane.addRow(5, new Label("bio "), UI\_edit\_person\_bio);

gPane.addRow(0, new Label("Relationship with "), UI\_edit\_existed\_person);

gPane.addRow(1, new Label("Relationship type "), UI\_edit\_person\_relation);

gPane.addRow(2, new Label(" (can be \"spouse/child/father/mother\" ) "));

gPane.addRow(6, Submit\_new\_person);

return gPane;

}

// action when click "add this new person to Tree"

private void doClick(Event evt) {

//for testing - shows what needed to add a new person

if (!"male".equals(UI\_edit\_person\_gender.getText())

&& !"female".equals(UI\_edit\_person\_gender.getText())) {

System.out.println("gender must be male or female");

}

if ("".equals(UI\_edit\_person\_firstname.getText())) {

System.out.println("a person must at least have a name");

}

//add this new person to tree data

person addingPerson = new person(UI\_edit\_person\_firstname.getText(),

UI\_edit\_person\_surname.getText(),

UI\_edit\_person\_gender.getText(),

UI\_edit\_person\_married\_surname.getText(),

UI\_edit\_person\_address.getText(),

UI\_edit\_person\_bio.getText());

personInterface.addPersonToTree(addingPerson);

//add new person according to their relation

if (!UI\_edit\_existed\_person.getText().equals("")) {

person existedPerson = personInterface.getPersonWithName(UI\_edit\_existed\_person.getText());

try {

if ("spouse".equals(UI\_edit\_person\_relation.getText())) {

existedPerson.addSpouse(addingPerson);

}

if ("child".equals(UI\_edit\_person\_relation.getText())) {

existedPerson.addChildren(addingPerson);

}

if ("father".equals(UI\_edit\_person\_relation.getText())) {

existedPerson.addFather(addingPerson);

}

if ("mother".equals(UI\_edit\_person\_relation.getText())) {

existedPerson.addMother(addingPerson);

}

} catch (Exception ex) {

System.out.println("Failed to add person to Tree's data " + ex);

}

}

reloadTreeView();

}

//here are where to make up the family tree -- quick tree view - for testing

private ArrayList<person> getSample() {

ArrayList<person> sample = new ArrayList<>();

person person1 = new person("tony", "junior", "male");

person person2 = new person("will", "myth", "male");

person person3 = new person("bob", "uncle", "male");

person person4 = new person("howard", "tark", "male");

person person5 = new person("jerry", "", "male");

person person6 = new person("elise", "", "female");

person person7 = new person("tom", "holl", "male");

person person8 = new person("may", "pot", "female");

person person9 = new person("maria", "Shark", "female");

person person0 = new person("peeper", "pott", "female");

person person10 = new person("yoda", "", "male");

sample.add(person1);

sample.add(person2);

sample.add(person3);

sample.add(person4);

sample.add(person5);

sample.add(person6);

sample.add(person7);

sample.add(person8);

sample.add(person9);

sample.add(person0);

person1.addFather(person4);

person1.addSpouse(person0);

person0.addSpouse(person1);// would this have any issue? seem not

person1.addChildren(person9);

person1.addChildren(person10);

person9.setBio("this this newly added bio");//editing existing people is okay

person0.addMother(person8);

person8.addChildren(person3);

person3.addChildren(person2);

person2.addSpouse(person6);

person6.addChildren(person7);

person6.addFather(person5);

return sample;

}

}

package as2ict373;

import java.util.ArrayList;

import javafx.scene.control.TreeItem;

/\*\*

\*

\* @author Vu Hoang Quan

\*/

//handle UI and I/O of person class

### public class personHandler {

private ArrayList<person> data;

public personHandler() {

data = new ArrayList<>();

}

//new person added to Tree must at least has a firstname

//and gender must be male or female

public void addPersonToTree(person newPerson) {

if (newPerson.getGender().equals("male")

|| newPerson.getGender().equals("female")) {

if (!newPerson.getFirstname().equals("")) {

data.add(newPerson);

}

}

}

// find person with matching name ( firstname/lastname/fullname with space)

public person getPersonWithName(String personName) {

for (person eachPerson : data) {

if (eachPerson.getFirstname().equals(personName)

|| eachPerson.getSurname().equals(personName)

|| personName.equals(eachPerson.getFirstname() + " " + eachPerson.getSurname())) {

return eachPerson;

}

}

return null;

}

//call familyTreeUI and return a nested TreeItem for UI

public TreeItem<person> generateFamilyTree(person person1) {

return this.familyTreeUI(person1, person1);

}

//this function is a helper for generateFamilyTree()

//return a Tree <person> of people firstname

//param: calling\_person is the trailing person and root is the current person in the recursive search

public TreeItem<person> familyTreeUI(person calling\_person, person root) {

TreeItem<person> itemRoot = new TreeItem<>(root);

if (root.getSpouse() != null && root.getSpouse() != calling\_person) {

boolean child\_calls = false;

for (person eachChild : root.getChildList()) {

if (eachChild == calling\_person) {

child\_calls = true;

break;

}

}

if (child\_calls == false) {

TreeItem<person> spouseList = new TreeItem<>(new person("Spouse: "));

spouseList.getChildren().add(familyTreeUI(root, root.getSpouse()));

itemRoot.getChildren().add(spouseList);

}

}

if (root.getFather() != null || root.getMother() != null) {

if (root.getFather() != calling\_person && root.getMother() != calling\_person) {

TreeItem<person> parentList = new TreeItem<>(new person("Parent:"));

if (root.getFather() != null) {

if (root.getFather() != calling\_person && root.getMother() != calling\_person) {

parentList.getChildren().add(familyTreeUI(root, root.getFather()));

}

}

if (root.getMother() != null) {

if (root.getMother() != calling\_person && root.getFather() != calling\_person) {

parentList.getChildren().add(familyTreeUI(root, root.getMother()));

}

}

itemRoot.getChildren().add(parentList);

}

}

if (!root.getChildList().isEmpty() && root.getSpouse() != calling\_person /\* && AllChildAreNotCallingPerson(calling\_person,root)==false\*/) {

//prevent showing "child:" with nothing in it

if (has1ChildAndIsCallingPerson(calling\_person, root) == false) {

TreeItem<person> childList = new TreeItem<>(new person("Child:"));

for (person eachPerson : root.getChildList()) {

if (eachPerson != calling\_person) {

childList.getChildren().add(familyTreeUI(root, eachPerson));

}

}

itemRoot.getChildren().add(childList);

}

} else {

//System.out.println("child list is null");

}

return itemRoot;

}

//return true if all childs are not the person called this function

private boolean has1ChildAndIsCallingPerson(person calling\_person, person root) {

if (root.getChildList().size() == 1 && root.getChildList().get(0) == calling\_person) {

return true;

} else {

return false;

}

}

public ArrayList<person> getData() {

return this.data;

}

public void setData(ArrayList<person> data) {

this.data = data;

}

}

package as2ict373;

/\*\*

\*

\* @author Vu Hoang Quan

\*/

import java.util.ArrayList;

import java.io.Serializable;

### public class person implements Serializable {

private String firstname = "unspecified";// not null

private String surname = "unspecified";

private String gender = "unspecified";// not null

private String surname\_after\_marriage = "unspecified";

private String address = "unspecified";

private String bio = "";

// may have a spouse or none

private person spousePerson;

// may have none or many childrens

private ArrayList<person> chilPersonsList = new ArrayList<person>();

private person fathePerson;// maybe unknown/empty

private person mothePerson;// maybe unknown/empty also

public person(String firstname, String surname, String gender) {

this.firstname = firstname;

this.surname = surname;

this.gender = gender;

}

//add a empty person - for Labeling the TreeView

public person(String labelText) {

this.bio = labelText;

}

// add a new person with full infomation

public person(String firstname, String surname, String gender, String surname\_after\_marriage, String address,

String bio) {

this.firstname = firstname;

this.surname = surname;

this.gender = gender;

this.surname\_after\_marriage = surname\_after\_marriage;

this.address = address;

this.bio = bio;

}

// print a person basic infomation - for testing only

public void printPersonInfo() {

System.out.print(" " + firstname + "|");

System.out.print(" " + surname + "|");

System.out.print(" " + gender + "|");

System.out.print(" " + surname\_after\_marriage + "|");

System.out.print(" " + address + "|");

System.out.print(" " + bio + "|");

System.out.println();

}

// add spouse > the other will have this person as their spouse as well

public void addSpouse(person newPerson) {

if (!newPerson.gender.equals(this.gender)) {

this.spousePerson = newPerson;

newPerson.spousePerson = this;

// if a person already have childs

if (!this.chilPersonsList.isEmpty()) {

// childrens will have same parent

for (person eachChild : chilPersonsList) {

eachChild.mothePerson = newPerson;

}

// then spouse will have the same children

newPerson.chilPersonsList = new ArrayList<person>(this.chilPersonsList);

}

}

}

// add father > that father will have this person as child

public void addFather(person father) {

if ("male".equals(father.gender)) {

this.fathePerson = father;

father.chilPersonsList.add(this);

} else {

System.out.println("person " + this.firstname + " father " + "must be male");

}

if (father.spousePerson != null) {

// add father's spouse as mother

this.mothePerson = father.spousePerson;

//mother will have this person as children

mothePerson.chilPersonsList.add(this);

}

}

// add mother > that mother will have this person as child

public void addMother(person mother) {

if ("female".equals(mother.gender)) {

this.mothePerson = mother;

mother.chilPersonsList.add(this);

} else {

System.out.println("person " + this.firstname + " mother " + "must be female");

}

}

// add children > that child will take this person as their parent

// (consider this person spouse as well)

public void addChildren(person child) {

if ("male".equals(this.gender)) {

child.fathePerson = this;

} else {

child.mothePerson = this;

}

if (this.spousePerson != null) {

if ("male".equals(this.gender)) {

child.fathePerson = this;

child.mothePerson = this.spousePerson;

} else {

child.fathePerson = this.spousePerson;

child.mothePerson = this;

}

spousePerson.chilPersonsList.add(child);

}

this.chilPersonsList.add(child);

}

public void setMarriedSurname(String marriedSurname) {

this.surname\_after\_marriage = marriedSurname;

}

public void setAddress(String address) {

this.address = address;

}

public void setBio(String bio) {

this.bio = bio;

}

public String getFirstname() {

return this.firstname;

}

public String getSurname() {

return this.surname;

}

public String getGender() {

return this.gender;

}

public String getMarriedSurname() {

return this.surname\_after\_marriage;

}

public String getAddress() {

return this.address;

}

public String getBio() {

return this.bio;

}

public person getSpouse() {

return this.spousePerson;

}

public person getFather() {

return this.fathePerson;

}

public person getMother() {

return this.mothePerson;

}

public ArrayList<person> getChildList() {

return chilPersonsList;

}

//override for TreeItem<person> to show text instead of person name

// example: clicked into "parent:" will show as "unspecified"

@Override

public String toString() {

if (firstname.equals("unspecified")) {

return bio;

} else {

return this.firstname + " " + this.surname;

}

}

}