Biuro matrymonialne

Dokumentacja

Wykonał
Patryk Wolski

1. Spis treści

2. Wymagania dotyczące projektu:

a. baza zawiera dane osobowe
b. oprócz danych osobowych baza ma pole zawierające słowa kluczowe (np. zainteresowania czy preferencje osób)
c. program ma dokonywać kojarzenia par korzystając z danych w bazie, wyszukiwać pary osób odpowiadające preferencjom
d. dane przechowywane w pliku tekstowym (zapis i odczyt)
e. przeglądanie zawartości bazy, wg różnych kryteriów np.. miejscowość, wiek, zainteresowania, preferencje
f. dodawanie rekordu na końcu bazy
g. uwzględnij klienta i pracownika biura

a. Baza danych osobowych musi zawierać wszystkie dane ściśle powiązane z konkretnymi osobami. Niektóre z nich są dostępne dla wszystkich osób(użytkowników). Użytkownik ma wgląd i możliwość edytowania swoich niektórych danych. Inne dane takie jak login czy data urodzenia są niezmienne i niedostępne. Jednocześnie baza musi aktualizować zmieniane dane tak, by edytujący oraz inni użytkownicy mogli te zmiany od razu zauważyć.
b. baza przechowuje pole które zawiera tekst w postaci dowolnego opisu ustalanego przy rejestracji z możliwością późniejszej jego edycji
c. program po wybraniu opcji kojarzenia w pary początkowo kojarzy osobę zalogowaną z osobami o przeciwnej preferencji. Jeśli osoba zalogowana jest mężczyzną program wyszukuje kobiety i odwrotnie. W oknie kojarzenia w pary do wyboru jest kwestia preferencji. Poza preferencjami brany pod uwagę jest podany przedział wiekowy. Jeśli skojarzona osoba przypadnie zalogowanej osobie do gustu może dodać ją do znajomych.
d. program zapisuje do pliku tekstowego wszystkie zmiany u zalogowanej osoby oraz dodaje do pliku zarejestrowane osoby
e. podczas wyszukiwania wypisywane są dane osób które spełniają w części lub całkowicie podane kryteria
f. dodawanie rekordu na końcu bazy następuje podczas pomyślnej rejestracji użytkownika, od tej pory figuruje on w bazie danych
g. klient ma możliwość zalogowania się po wcześniejszej rejestracji. Naturalnie nie musi się rejestrować przed każdym zalogowaniem. Pracownik biura ma przypisane do siebie dane logowania, program sprawdza czy logowana osoba podała login i hasło pracownika, jeśli tak wyświetlany jest panel z wyświetloną całą bazą danych z możliwościa usunięcia wybranej osoby z bazy danych.

(dodatkowo). opcja losowania dowolnej osoby z całej bazy danych z możliwością dodania do znajomych oraz możliwość edycji niektórych danych zalogowanej osoby.

3. Opis algorytmów, zmiennych

Załadowanie bazy danych do programu.

```
public void refresh(){
        this.listaKlientow.clear();

        Klient klient;

        String zdanie, login, haslo, imie, nazwisko, miasto, malefemale, tagi, przyjaciele;
        int dzien, miesiac, rok;
        String[] parts;

        String filePath = "zapis.txt";
        BufferedReader fileReader = null;

        try {
        fileReader = new BufferedReader(new FileReader(filePath));

        while ((zdanie = fileReader.readLine()) != null) {
            parts = zdanie.split("-");
            login = parts[0];
            haslo = parts[1];
            imie = parts[2];
            nazwisko = parts[3];
            dzien = Integer.parseInt(parts[4]);
            miesiac = Integer.parseInt(parts[5]);
            rok = Integer.parseInt(parts[6]);
            miasto = parts[7];
            malefemale = parts[8];
            tagi = parts[9];
            przyjaciele = parts[10];

            klient = new Klient(login, haslo, imie, nazwisko, dzien, miesiac, rok, miasto, malefemale, tagi, przyjaciele);
            this.listaKlientow.add(klient);
            klient.setBaza(this);
            klient.setPrzyjaciele();
        };
    }
            catch (FileNotFoundException ex) {
```

Klasa Baza zawiera zmienną “listaKlientow” typu ArrayList<Klient> reprezentującą całą bazę danych osób zarejestrowanych. Obiekt klasy Klient reprezentuje klienta. Na początku tworzony jest strumień do pliku tekstowego z danymi oraz odczytywanie z pliku każdej linii do momentu gdy napotkany jest koniec pliku.
Podczas odczytywania linni, linia zapisywana jest do zmiennej “zdanie” typu String, następnie String “zdanie” jest dzielony na tablicę String[] “parts” złożoną z 10 elementów. Każdy z nich reprezentuje inne dane zarejestrowanej osoby. Następnie zostaje stworzony obiekt klasy Klient z parametrami pobranymi z pliku a następnie dodawany do ArrayListy “listaKlientow”. Do utworzonego klienta zostaje również przekazana obecna baza na podstawie której obiekt “klient” jest wstanie odczytać pozostałe informacje.

Klasa Klient

```
public class Klient extends Osoba{

    final String login;
    private String haslo;
    private String tagi;
    private String malefemale;
    private ArrayList<Klient> friendList = new ArrayList<Klient>();
    private Baza baza;
    private String przyjaciele;

    Klient(String login, String haslo, String imie, String nazwa,
            int dzien, int miesiac, int rok, String miasto,
            String malefemale, String tagi,String przyjaciele){

        super(imie, nazwa, dzien, miesiac, rok, miasto);
        this.login = login;
        this.haslo = haslo;
        this.tagi = tagi;
        this.malefemale = malefemale;
        this.baza = baza;
        this.przyjaciele=przyjaciele;
    }
```

Podczas tworzenia obiektu klasy Klient trzeba przekazać odpowiednie parametry, które są przypisywane do tego obiektu. Klasa Klient dziedziczy po abstrakcyjnej klasie Osoba, która posiada pola takie jak imie, nawisko itd.

Klasa Osoba

```
public abstract class Osoba {
    private String imie;
    private String nazwa;
    final int dzien;
    final int miesiac;
    final int rok;
    private int age;
    private String miasto;
    private Calendar calendar;
    private SimpleDateFormat dateFormat;

    /**
     *
     * @param imie
     * @param nazwa
     * @param dzien
     * @param miesiac
     * @param rok
     * @param miasto
     */
    public Osoba(String imie, String nazwa, int dzien, int miesiac, int rok, String miasto) {
        this.imie = imie;
        this.nazwa = nazwa;
        this.dzien = dzien;
        this.miesiac = miesiac;
        this.rok = rok;
        this.miasto = miasto;
    }
```

```
public int getAge(){

        calendar = Calendar.getInstance();
        dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");

        int nowDay = calendar.get(Calendar.DAY_OF_MONTH);
        int nowMonth = calendar.get(Calendar.MONTH);
        int nowYear = calendar.get(Calendar.YEAR);

        int age = nowYear-rok;

        if (nowMonth < miesiac) {
            age--;
        } else if ((nowMonth == miesiac) && (nowDay < dzien)) {
            age--;
            if (age < 0) {
                age = 0;
            }
        }

    return age;
}
```

Metoda getAge() zwraca wartość zmiennej “age” typu int reprezentującą wiek zarejestrowanego klienta na podstawie pobranych wartości pól klasy oraz daty występującej w momencie wywołania metody.

Pozyskanie pozostałych informacji dla obiektu klasy Klient

```
public void setPrzyjaciele(){
    this.friendList.clear();

    String[] parts;
    parts = this.przyjaciele.split("/");

    for(int i=0; i < parts.length ; i++){
        for(int j=0; j < this.baza.sizeListaKlientow(); j++){
            if(parts[i].equals(this.baza.getListaKlientow(j).getLogin()) && !parts[i].toLowerCase().contains(this.przyjaciele)){
                addFriendList(this.baza.getListaKlientow(j));
            }
        }
    }
}
```

Obiekt klasy Klient posiada pole “przyjaciele” typu String który przechowuje informacje o dodanych do przyjaciół innych osób z bazy. String “przyjaciele” dzielony jest na części i przypisywany do tablicy Stringów “parts” a następnie każdy element tablicy “parts” zawierający loginy dodanych jest porównywany z loginami osób w bazie. Jeśli taki login występuje w bazie zostaje pobierany obiekt klasy Klient z bazy wszystkich obiektów tej klasy “listaKlientow” za pomocą metody get.ListaKlientow(int i), która zwraca obiekt klasy Klient z ArrayListy “listaKlientow” o podanym indeksie w parametrze. Następnie pobrany obiekt jest przekazywany w parametrze do metody addFierdndList(Klient klient) która dodaje do pola “friendList” typu ArrayList<Klient> reprezentującego listę przyjaciół danego klienta.

Z tak pobraną bazą danych program jest gotowy by móc zarejestrować bądź zalogować klienta.

Rejestracja

Klasa NewFrame dziedziczy po klasie JFrame i jest oknem rejestracji. Wpisywane są tu dane nowego klienta. Po naciśnięciu przycisku sprawdzane są wpisane dane.

```
utw.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(imie.getText().length()<3 ||
           imie.getText().length()>15 ||
           imie.getText().contains(" ") ||
           Pattern.compile( "[0-9]" ).matcher( imie.getText() ).find()){

            imie.setBorder(BorderFactory.createLineBorder(Color.red));
            shouldBreak = true;
        }
        else{
            imie.setBorder(defaultBorder);
        }

        if(nazwa.getText().length()<3 || nazwa.getText().length()>15){
            nazwa.setBorder(BorderFactory.createLineBorder(Color.red));
            shouldBreak = true;
        }
        else{
            nazwa.setBorder(defaultBorder);
        }

        if(login.getText().length()<3 || login.getText().length()>15 || login.getText().contains(" ")){
            login.setBorder(BorderFactory.createLineBorder(Color.red));
            shouldBreak = true;
        }
        else{
            login.setBorder(defaultBorder);
        }

        if(haslo.getText().length()<3 || haslo.getText().length()>15){
            haslo.setBorder(BorderFactory.createLineBorder(Color.red));
            shouldBreak = true;
        }
        else{
            haslo.setBorder(defaultBorder);
        }
```

Sprawdzane są między innymi czy podane dane są zbyt długie bądź krótkie, czy takie dane jak imię czy nazwisko posiadają cyfry bądź spacje.

```
for(int i=0; i<baza.sizeListaKlientow(); i++){
    if(login.getText().equals(baza.getListaKlientow(i).getLogin())){
        login.setBorder(BorderFactory.createLineBorder(Color.red));
        shouldBreak = true;
        break;
    }

    else if(nazwa.getText().equals(baza.getListaKlientow(i).getNazwa())){
        nazwa.setBorder(BorderFactory.createLineBorder(Color.red));
        shouldBreak = true;
        break;
    }
}
```

Pętla for sprawdza czy podany login lub nazwa użytkownika figuruje w bazie danych.

```
dateValidator = new DateValidator();

if (dateValidator.isThisDateValid(
        day.getText() + "/"
        + month.getText() + "/"
        + year.getText(), "dd/MM/yyyy"
)) {
    day.setBorder(defaultBorder);
    month.setBorder(defaultBorder);
    year.setBorder(defaultBorder);
} else {

    day.setBorder(BorderFactory.createLineBorder(Color.red));
    month.setBorder(BorderFactory.createLineBorder(Color.red));
    year.setBorder(BorderFactory.createLineBorder(Color.red));
    shouldBreak = true;
}
```

Za pomocą obiektu klasy DateValidator sprawdzana jest wprowadzona data. Czy został podany 13 miesiąc bądź podczas podania miesiąca lutego został podany również 31 dzień itp. Metoda isThisDateValid() sprawdza czy z podanej daty uda się stworzyć obiekt klasy Date, jeśli tak metoda zwraca wartość true, jeśli jakiś warunek nie zachodzi zwraca false. Jeśli zwróci wartość true oznacza to, że podana data jest poprawna.

```
int nowDay = calendar.get(Calendar.DAY_OF_MONTH);
int nowMonth = calendar.get(Calendar.MONTH);
int nowYear = calendar.get(Calendar.YEAR);
int age = nowYear - Integer.parseInt(year.getText());

if (nowMonth < Integer.parseInt(month.getText())) {
    age--;
} else if ((nowMonth == Integer.parseInt(month.getText())) && (nowDay < Integer.parseInt(day.getText()))) {
    age--;
    if (age < 0) {
        age = 0;
    }
}

if(age >= 18){
    day.setBorder(defaultBorder);
    month.setBorder(defaultBorder);
    year.setBorder(defaultBorder);
}
else{
    day.setBorder(BorderFactory.createLineBorder(Color.red));
    month.setBorder(BorderFactory.createLineBorder(Color.red));
    year.setBorder(BorderFactory.createLineBorder(Color.red));
    shouldBreak = true;
}

if(Integer.parseInt(year.getText())<1920){
    year.setBorder(BorderFactory.createLineBorder(Color.red));
    shouldBreak = true;
}
else{
    year.setBorder(defaultBorder);
}
```

Następnie obliczany jest wiek na podstawie podanych danych i datą w momencie rejestracji. Rejestrowana osoba musi mieć powyżej 18 lat a minimalna wartość roku urodzenia to 1920.

```
if(!shouldBreak){
    przyjaciele=" ";

    try {
        FileWriter zapis = null;
        zapis = new FileWriter("zapis.txt", true);
        zapis.write(
                login.getText()+"-"+
                haslo.getText()+"-"+
                imie.getText()+"-"+
                nazwa.getText()+"-"+
                Integer.parseInt(day.getText())+"-"+
                Integer.parseInt(month.getText())+"-"+
                Integer.parseInt(year.getText())+"-"+
                miasto.getText()+"-"+
                group.getSelection().getActionCommand()+"-"+
                tag.getText().replaceAll("\n", " ")+"-"+
                przyjaciele+"\r\n"
        );
        zapis.close();
    }

    catch (FileNotFoundException ex) {
        Logger.getLogger(NewFrame.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(NewFrame.class.getName()).log(Level.SEVERE, null, ex);
    }

    dispose();
    baza.refresh();
    f = new OldFrame(baza);
    f.setVisible(true);

} else {
    shouldBreak=false;
}
```

Jeśli któryś warunek zwróci “true” pozycja z której pobrana wartość spowodowała spełnienie warunku zostanie podświetlona na czerwono a zmienna “shouldBrake” typu boolean zmieni wartość na true. Jeśli tak się stanie oznacza to, że wprowadzane dane były nieprawidłowe i klient nie zostanie zarejestrowany. Jeśli “shouldBrake” zwróci false oznacza to, że dane są prawidłowe i można pomyślnie zarejestrować klienta dopisując jego dane na koniec bazy danych i pliku tekstowego. Domyślnie nowo zarejestrowany użytkownik nie posiada przyjaciół z portalu zatem wartość zmiennej przyjaciele posiada wartość “ “. Jeśli dane zostaną zapisane pomyślnie obecne okno rejestracji zostaje zamknięte, baza zostaje odświeżona i przekazana do nowego obiektu Klasy OldFrame, który jest oknem logowania. Okno rejestracji zostaje zamknięte.

Logowanie

```
zaloguj.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

    if(login.getText().equals("admin")){
            if(haslo.getText().equals("admin")){
                f = new Admin(baza);
                dispose();
                f.setVisible(true);
            }
    }

    for(int i=0; i<baza.sizeListaKlientow(); i++){
        if(login.getText().equals(baza.getListaKlientow(i).getLogin())){
            if(haslo.getText().equals(baza.getListaKlientow(i).getHaslo())){
                f = new ProfileFrame(baza, baza.getListaKlientow(i));
                dispose();
                f.setVisible(true);
            }
        }
    }

    info2.setForeground(Color.red);
    info2.setText("Niepoprawny login lub hasło.");

} } );
```

Program sprawdza czy podanymi danymi logowania są danymi administratora, jeśli tak otwierany jest panel administratora.
W pętli for sprawdzane jest czy podany login istnieje, jeśli tak to sprawdzane jest czy podane hasło jest hasłem przypisanym do tego loginu. Jeśli nie wyświetlany jest komunikat, jeśli tak zostaje utworzony nowy obiekt klasy ProfileFrame który jest menu głównym zalogowanej osoby. Do konstruktora obiektu przekazywana jest obecna baza danych oraz obiekt Klasy klient którego dane zostały podane w danych logowania. Obecne okienko zostaje zamknięte.

Menu główne

```
losuj.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(l==null || !l.isVisible()){
            l = new RandomProfileFrame(baza, klient);
            l.setVisible(true);
            dispose();
        }
} } );

para.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(p==null || !p.isVisible()){
            p = new PairFrame(baza, klient);
            p.setVisible(true);
            dispose();
        }
} } );

szukaj.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(s==null || !s.isVisible()){
            s = new SearchFrame(baza,klient);
            s.setVisible(true);
            dispose();
        }
} } );
```

```
edit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(ed==null || !ed.isVisible()){
            ed = new EditFrame(baza,klient);
            ed.setVisible(true);
            dispose();
        }
} } );

friends.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(fr==null || !fr.isVisible()){
            fr = new FriendsFrame(baza,klient);
            fr.setVisible(true);
        }
} } );
wyjdz.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        f = new MainFrame();
        if(fr!=null){
            fr.dispose();
        }
        if(l!=null){
            l.dispose();
        }
        if(p!=null){
            p.dispose();
        }
        if(s!=null){
            s.dispose();
        }
        if(ed!=null){
            ed.dispose();
        }
        dispose();
} } );
```

Jeśli któryś przycisk zostanie naciśnięty sprawdzane jest czy obiekt, który ma zostać utworzony nie istnieje lub czy jest on niewidoczny. Jeśli warunek jest spełniony tworzony jest dany obiekt i wyświetlany. Jeśli zostanie naciśnięty przycisk “wyjdz” zostaje utworzony nowy obiekt MainFrame a jeśli jednocześnie inne obiekty są utworzone zostają wyczyszczone.

Losowanie

```
los = r.nextInt(((baza.sizeListaKlientow())));

imie.setText(baza.getListaKlientow(los).getImie());
nazwa.setText(baza.getListaKlientow(los).getNazwa());
wiek.setText(Integer.toString(baza.getListaKlientow(los).getAge()));
miasto.setText(baza.getListaKlientow(los).getMiasto());
tagi.setText(baza.getListaKlientow(los).getTagi());

losuj.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        dispose();
        l = new RandomProfileFrame(baza, klient);
        l.setVisible(true);
} } );

dodaj.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < baza.sizeListaKlientow(); i++) {
            if (nazwa.equals(baza.getListaKlientow(i).getNazwa())) {
                klient.addPrzyjaciele(baza.getListaKlientow(i));
                klient.setPrzyjaciele();
                baza.refresh();
                klient.refresh();
            }
        }
} } );

wyjdz.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        dispose();
        f = new ProfileFrame(baza, klient);
        f.setVisible(true);
} } );
```

Po utworzeniu obiektu losowana jest liczba od 0 do liczby obiektów w bazie danych i zapisywana do zmiennej “los”. Następnie wyświetlane zostają dane obiektu o indeksie “los” z bazy danych. Zalogowana osoba ma w tym momencie dwie możliwości losowania osób dalej i wtedy tworzony jest identyczny obiekt a obecny zamykany lub dodanie wyświetlanej osoby do listy przyjaciół. W drugim przypadku w pętli for wyszukiwana jest wyświetlana osoba w bazie, pobierany indeks oraz wyszukany obiekt klasy Klient dodawany do listy przyjaciół zalogowanej osoby. Dane zostają zapisane do bazy oraz zalogowanego klienta.

Dodawanie przyjaciół

```
public void addPrzyjaciele(Klient klient) {
    if(!this.przyjaciele.toLowerCase().contains(klient.getNazwa().toLowerCase())){
        this.przyjaciele=this.przyjaciele+"/"+klient.getLogin();
    }


    String sentence, filePath;
    String[] parts;
    ArrayList<String> tempArray = new ArrayList<>();


            filePath = "zapis.txt";
            FileReader fileReader = null;
            Scanner reader = null;

            try {
                fileReader = new FileReader(filePath);
                reader = new Scanner(fileReader);

            while(reader.hasNextLine()){
                sentence = reader.nextLine();
                parts = sentence.split("-");

                if(parts[0].equals(this.getLogin())){
                    tempArray.add(
                        this.getLogin() + "-" +
                        this.getHaslo() + "-" +
                        this.getImie() + "-" +
                        this.getNazwa() + "-" +
                        this.getDzien() + "-" +
                        this.getMiesiac() + "-" +
                        this.getRok() + "-" +
                        this.getMiasto() + "-" +
                        this.getMalefemale() + "-" +
                        this.getTagi() + "-" +
```

metoda addPrzyjaciele() dodaje do pola “przyjaciele” typu String separator ”/” oraz login dodanej osoby do przyjaciół jeśli dodawana osoba nie jest jeszcze zapisana w polu “przyjaciele” a następnie baza danych w pliku tekstowym zostaje zapisana

Kojarzenie w pary

```
public void ile() {
    ile.clear();
    if(jMinAge.getText().equals("") && !jMaxAge.getText().equals("")){
                maxAge = Integer.parseInt(jMaxAge.getText());
            }
            else if(jMaxAge.getText().equals("") && !jMinAge.getText().equals("")){
                minAge = Integer.parseInt(jMinAge.getText());
            }
            else if(!jMinAge.getText().equals("") && !jMaxAge.getText().equals("")){
                maxAge = Integer.parseInt(jMaxAge.getText());
                minAge = Integer.parseInt(jMinAge.getText());
            }
            else{}

    if (group.getSelection().getActionCommand() != null) {
        for (int i = 0; i < baza.sizeListaKlientow(); i++) {
            if (
                group.getSelection().getActionCommand().equals(baza.getListaKlientow(i).getMalefemale()) &&
                ((jMinAge.getText().equals("") && jMaxAge.getText().equals("")) || (jMinAge.getText().equals("") &&
                 baza.getListaKlientow(i).getAge()<=maxAge) || (minAge<=baza.getListaKlientow(i).getAge() &&
                 baza.getListaKlientow(i).getAge()<=maxAge) || (minAge<=baza.getListaKlientow(i).getAge() &&
                 jMaxAge.getText().equals(""))
                    )){
                ile.add(i);
            } else {
            }
        }
    }
    this.pom = 0;
}
```

Podczas tworzenia obiektu klasy PairFrame wywoływana jest metoda ile(). Metoda ta wyszukuje w bazie obiekty spełniające warunki(czy płeć jest zgodna z wybranym kryterium oraz czy wiek mieści się w podanym przez użytkownika przedziale. Gdy warunek jest spełniony do zmiennej “ile” typu ArrayListy<int> zapisany jest indeks wyszukanej osoby. Wywołana metoda czyści zmienną “ile” przed ponownym zliczaniem obiektów.

```
public void next() {
    if(ile.size()==0){
        nick = "";
        imie.setText("");
        nazwa.setText("");
        wiek.setText("");
        miasto.setText("");
        tagi.setText("");
    }
    if (pom < ile.size()) {
        nick = baza.getListaKlientow(ile.get(pom)).getNazwa();

        imie.setText(baza.getListaKlientow(ile.get(pom)).getImie());
        nazwa.setText(baza.getListaKlientow(ile.get(pom)).getNazwa());
        wiek.setText(Integer.toString(baza.getListaKlientow(ile.get(pom)).getAge()));
        miasto.setText(baza.getListaKlientow(ile.get(pom)).getMiasto());
        tagi.setText(baza.getListaKlientow(ile.get(pom)).getTagi());

        pom++;
    } else {
        pom = 0;
    }
}
```

Metoda next() wyświetla dane osoby z bazy danych o indeksie pobranym ze zmiennej “ile” gdzie zapisane były indeksy wyszukanych osób. Na początku sprawdzany jest warunek czy lista znalezionych osób jest zerem, jeśli tak to czyści wyświetlane dane. Następnie czy zmienna “pom” typu int będąca licznikiem wywołania metody next() jest mniejsza niż ilość zapisanych danych w zmiennej “ile”. Jeśli warunek jest spełniony zostają wyświetlane dane obiektu o indeksie zapisanym w zmiennej “ile” na indeksie “pom”. Jeśli warunek nie zostanie spełniony licznik “pom” zostaje wyzerowany co pozwala na wyświetlanie wyszukanych osób od początku.

```
para.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        if(!(actionCommand.equals(group.getSelection().getActionCommand())) ||
            (!jMinAge.getText().equals(minAgeText)) ||
            (!jMaxAge.getText().equals(maxAgeText))){
            actionCommand=group.getSelection().getActionCommand();
            minAgeText=jMinAge.getText();
            maxAgeText=jMaxAge.getText();
            getPairFrame().ile();
            getPairFrame().next();
        }
        else{
            getPairFrame().next();
        }

} } );
```

Podczas tworzenia obiektu klasy PairFrame zapisywana zostaje do zmiennej “actionCommand” typu String wartość zaznaczonego komponentu klasy JRadioButton oznaczającego płeć szukanej osoby oraz do zmiennych “maxAgeText” i “minAgeText”. Po naciśnięciu przycisku “para” sprawdzany jest czy zmienna “actionCommand”, “maxAgeText” oraz “minAgeText” mają wartości takie same jak na nowo pobrane wartości z odpowiednich komponentów. Jeśli są takie same wywoływana jest metoda next() na obiekcie klasy PairFrame. Jeśli są różne do zmiennych “actionCommand”, “maxAgeText” oraz “minAgeText” przypisywane są nowe wartości a następnie wywoływana metoda ile() oraz next(). Warunek ten pozwala na dynamiczną zmianę płci oraz wieku kojarzonych osób.

Wyszukiwanie

```
search.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        clerFoundKlient();

        name = jName.getText();
        nick = jNick.getText();

        if(jMinAge.getText().equals("") && !jMaxAge.getText().equals("")){
            maxAge = Integer.parseInt(jMaxAge.getText());
        }
        else if(jMaxAge.getText().equals("") && !jMinAge.getText().equals("")){
            minAge = Integer.parseInt(jMinAge.getText());
        }
        else if(!jMinAge.getText().equals("") && !jMaxAge.getText().equals("")){
            maxAge = Integer.parseInt(jMaxAge.getText());
            minAge = Integer.parseInt(jMinAge.getText());
        }
        else{}

        city = jCity.getText();

        try{
            malefemale = group.getSelection().getActionCommand();
        }
        catch(NullPointerException f){
            malefemale = "";
        }

        tags = jTag.getText();
```

Po naciśnięciu przycisku “search” czyszczona jest zmienna “foundKlient” typu Arraylist<Klient> która przechowuje wyszukane obiekty za pomocą metody clearFoundKlient(). Następnie pobierane są wpisane wartości do wyszukania wśród

obiektów. zmienna int nie może być “nullem” zatem jeśli nic nie jest wpisane w polu jMaxAge i jMinAge do zmiennej minAge i maxAge nie jest przypisywana żadna wartość. Również gdy nie zostanie wybrany komponent klasy JRadioButton do zmiennej typu String przypisywana jest wartość “”.

```
    for(int i=0; i<baza.sizeListaKlientow(); i++){
        if(name.equals("") || baza.getListaKlientow(i).getImie().toLowerCase().contains(name.toLowerCase())){
            if(nick.equals("") || baza.getListaKlientow(i).getNazwa().toLowerCase().contains(nick.toLowerCase())){
                if((jMinAge.getText().equals("") && jMaxAge.getText().equals("")) || (jMinAge.getText().equals("") &&
                        baza.getListaKlientow(i).getAge()<=maxAge) || (minAge<=baza.getListaKlientow(i).getAge() &&
                        baza.getListaKlientow(i).getAge()<=maxAge) || (minAge<=baza.getListaKlientow(i).getAge() &&
                        jMaxAge.getText().equals(""))){
                    if(city.equals("") || baza.getListaKlientow(i).getMiasto().toLowerCase().equals(city.toLowerCase())){
                        if(malefemale.equals("") || baza.getListaKlientow(i).getMalefemale().equals(malefemale)){
                            if(tags.equals("") || baza.getListaKlientow(i).getTagi().toLowerCase().contains(tags.toLowerCase())){
                                addFoundKlient(baza.getListaKlientow(i));
                            }
                        }
                    }
                }
            }
        }
    }

panel.repaint();
panel.refresh();
```

W pętli for wyszukiwane są obiekty które spełniają podane wcześniej warunki. Kaskadowo zostają sprawdzane warunki. Jeśli jeden warunek się spełni sprawdzany jest kolejny itd. Dla wygodnego wyszukiwania jeśli nie jest nic wpisane w odpowiednim polu JTextField np. name.equals(“”) zwraca true co pozwala na sprawdzanie kolejnego zagnieżdżonego warunku. Ponadto w bazie nie jest wyszukiwane wśród obiektów dokładnie to co jest wpisane tylko czy dany obiekt zawiera w swojej odpowiedniej danej wpisaną wartość. Np. gdy wpiszemy “a” w polu imię do zmiennej “foundKlient” zostaną dodane te obiekty, które w swojej danej “imie” posiadają wpisany wcześniej napis “a” oraz spełniają pozostałe warunki. Po wyszukaniu wszystkich obiektów zostaje wywołana metoda repaint() oraz refresh() na obiekcie klasy Found dziedziczącym po JPanel, który jest odpowiedzialny za wyświetlanie wyszukanych obiektów. Jest on dodany do wyświetlanego obiektu klasy SearchFrame dziedziczącym po JFrame, który jest oknem wyszukiwania osób.

```
public void refresh() {

    DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
    Object rowData[] = new Object[6];

    while (model.getRowCount() > 0) {
        for (int i = 0; i < model.getRowCount(); i++) {
            model.removeRow(i);
        }
    }

    for (int i = 0; i < foundKlient.size(); i++) {
        rowData[0] = foundKlient.get(i).getNazwa();
        rowData[1] = foundKlient.get(i).getImie();
        rowData[2] = foundKlient.get(i).getAge();
        rowData[3] = foundKlient.get(i).getMiasto();
        rowData[4] = foundKlient.get(i).getPolish();
        rowData[5] = foundKlient.get(i).getTagi();
        model.addRow(rowData);
    }

}
```

Metoda refresh() na początku czyści wszystkie wiersze wypisane w tabeli która znajduje się na tym obiekcie. Następnie wywoływana metoda na obiekcie klasy Found pobiera dane ze zmiennej “foundKlient” typu ArrayList<Klient> przekazanej w konstruktorze oraz w pętli “for” przypisuje do każdej kolumny danego wiersza dane ze zmiennej “foundKlient”. Następnie dany wiersz jest wyświetlany w tabeli.

```
private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
    int index = jTable1.getSelectedRow();
    TableModel model = jTable1.getModel();
    String nick = model.getValueAt(index, 0).toString();

    for (int i = 0; i < baza.sizeListaKlientow(); i++) {
        if (nick.equals(baza.getListaKlientow(i).getNazwa())) {
            if (f == null || !f.isVisible()) {
                open=true;
                f = new AddProfileFrame(baza, klient, baza.getListaKlientow(i),open);
                f.setVisible(true);
            }
        }
    }
}
```

W tabeli zostają wyświetlane dane obiektów, które zostały wyszukane. Gdy klikniemy na dany wiersz reprezentujący obiekt zostaje pobrany indeks kliknięty wiersza oraz do zmiennej “nick” typu String zostaje przypisana wartość pierwszej kolumny zaznaczonego wiersza. Następnie w bazie wyszukiwany jest obiekt którego zmienna “nazwa” jest taka sama z tą zapisaną w zmiennej “nick” a następnie zostaje

wyświetlony nowy obiekt klasy AddProfileFrame, z przekazaniem bazy danych, zalogowanego klienta, wybranego klienta oraz zmienna “open” boolean która przechowuje informację czy obiekt tej klasy został otwarty wcześniej.

```
dodaj.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {

        zalogowany.addPrzyjaciele(klient);
        zalogowany.refresh();
        zalogowany.setPrzyjaciele();
        baza.refresh();

        if((frame==null || !frame.isVisible()) && open()==false){
            frame = new AddProfileFrame(baza, zalogowany, klient, open());
            frame.setVisible(true);
        }
        dispose();
} } );
```

Gdy klikniemy dodaj na wyświetlonym obiekcie klasy AddProfileFrame do zalogowanego klienta zostaje dodany wyświetlony klient jako przyjaciel oraz baza danych zostaje zaktualizowana.

Wyświetlanie dodanych przyjaciół

```
add(panel = new Friends(baza, klient));
panel.setBounds(6, 10, 580, 220);

add(delete = new JButton("Usuń"));
delete.setBounds(198, 300, 100, 30);

add(exit = new JButton("Cofnij"));
exit.setBounds(302, 300, 100, 30);

panel.repaint();
panel.refresh(baza, klient);

delete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {


        for (int i = 0; i < baza.sizeListaKlientow(); i++) {
            if (panel.getNick().equals(baza.getListaKlientow(i).getNazwa())) {
                klient.delPrzyjaciele(baza.getListaKlientow(i));
                baza.refresh();
                klient.refresh();
                klient.setPrzyjaciele();
                panel.refresh(baza,klient);
                panel.repaint();


            }
        }

} } );
```

Do utworzonego obiektu klasy FriendsFrame przypisany zostaje obiekt klasy Friends dziedziczący po klasie JPanel zawierający tabelę. Po kliknięciu przycisku “delete” zaznaczony obiekt w tabeli zostaje usunięty z listy przyjaciół zalogowanego użytkownika, jego lista znajomych, baza danych, plik tekstowy oraz obiekt “panel” klasy Friends zostają zaktualizowane.

```
public void refresh(Baza baza, Klient klient){

    this.baza=baza;
    this.klient=klient;


    DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
    Object rowData[] = new Object[6];

    while (model.getRowCount() > 0) {
        for (int i = 0; i < model.getRowCount(); i++) {
            model.removeRow(i);
        }
    }

    for (int i = 0; i < klient.sizefriendList(); i++) {
            rowData[0] = klient.getPrzyjaciele(i).getNazwa();
            rowData[1] = klient.getPrzyjaciele(i).getImie();
            rowData[2] = klient.getPrzyjaciele(i).getAge();
            rowData[3] = klient.getPrzyjaciele(i).getMiasto();
            rowData[4] = klient.getPrzyjaciele(i).getPolish();
            rowData[5] = klient.getPrzyjaciele(i).getTagi();
            model.addRow(rowData);
        }
}
```

Metoda refresh() wywołana na obiekcie klasy Friends czyści tabelę przypisaną do tego obiektu a następnie wyświetla wszystkich przyjaciół zalogowanego użytkownika podobnie jak w obiekcie klasy Found.

Edycja danych

Obiekt klasy EditFrame pobiera dane zalogowanego klienta oraz wyświetla je w polu do wpisania nowych danych. Po naciśnięciu przycisku “edit” pobierane są wpisane wartości oraz analizowane oraz zapisane w pliku identycznie jak w przypadku rejestracji.

```
if(oldHaslo.getText().contains(klient.getHaslo()) || oldHaslo.getText().contains("")){
    oldHaslo.setBorder(defaultBorder);
    if(!newHaslo1.getText().contains("") && newHaslo1.getText().length()<3 || newHaslo1.getText().length()>15){
        newHaslo1.setBorder(BorderFactory.createLineBorder(Color.red));
        shouldBreak = true;
    }
    else{
        newHaslo1.setBorder(defaultBorder);
        if(newHaslo2.getText().contains(newHaslo1.getText()) || newHaslo2.getText().contains("")){
            newHaslo2.setBorder(defaultBorder);
            if(oldHaslo.getText().contains("") && newHaslo1.getText().contains("") && newHaslo2.getText().contains("")){
                changeHaslo=false;
            }
        }
        else{
            newHaslo2.setBorder(BorderFactory.createLineBorder(Color.red));
            shouldBreak = true;
        }
    }
}
else{
    oldHaslo.setBorder(BorderFactory.createLineBorder(Color.red));
    shouldBreak = true;
}
```

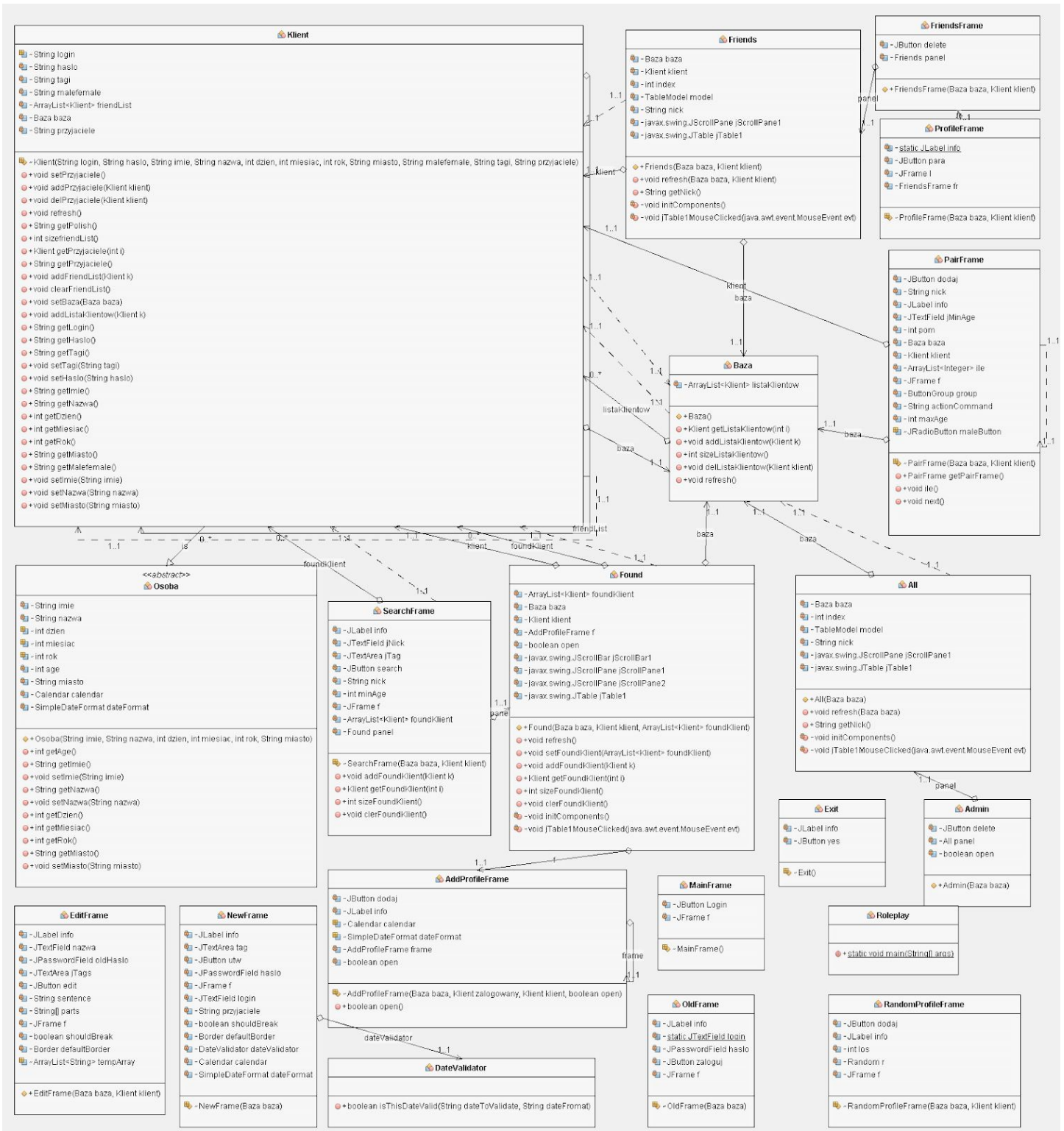
Dodatkowo sprawdzane jest czy podczas edytowania hasła wpisane stare hasło jest poprawne, czy nowe hasło spełnia warunki tak jak przy rejestracji oraz czy ponowne wpisanie nowego hasła dla potwierdzenia jest identyczne jak wpisane wcześniej nowe hasło.

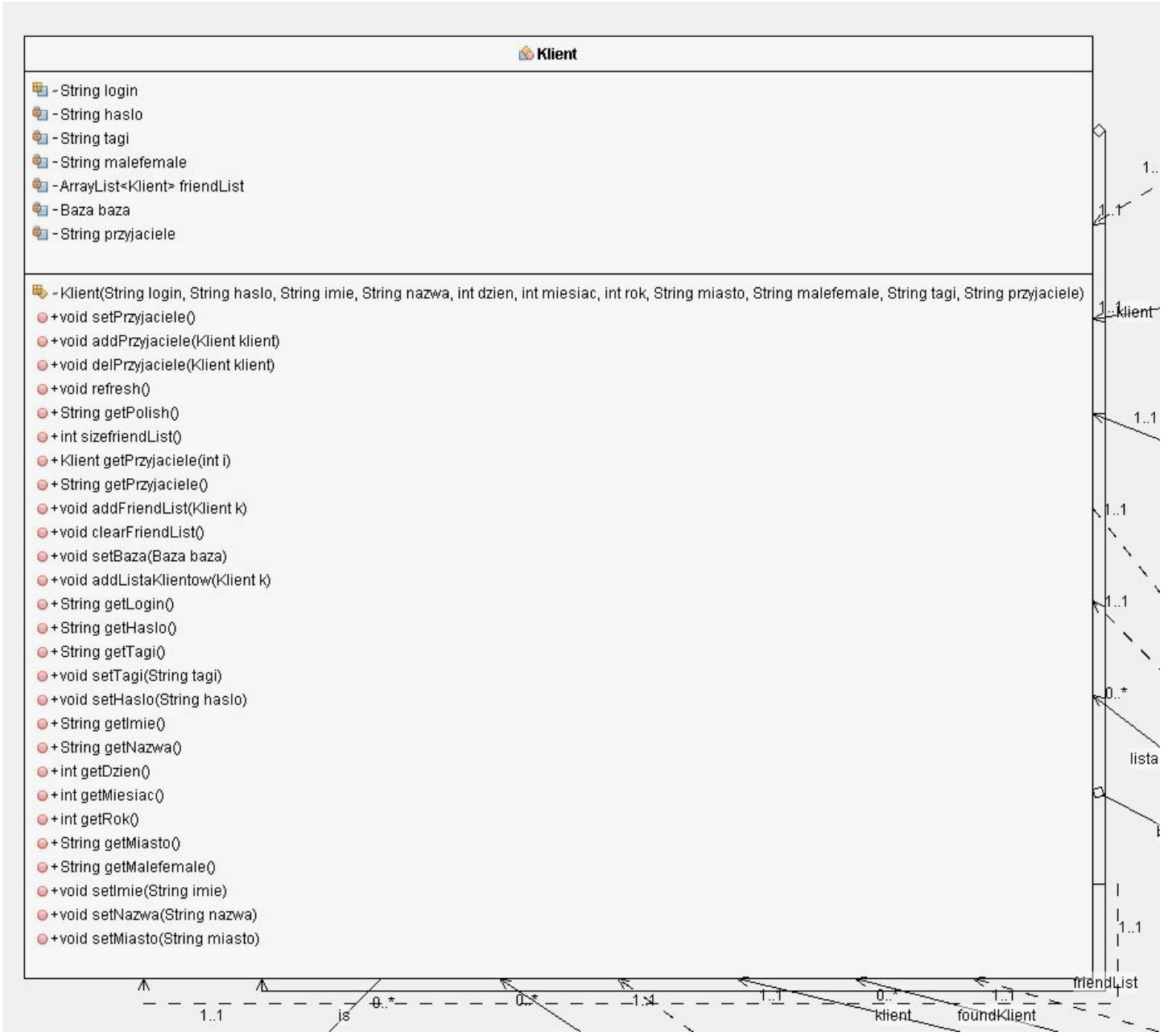
Zabezpieczenia.

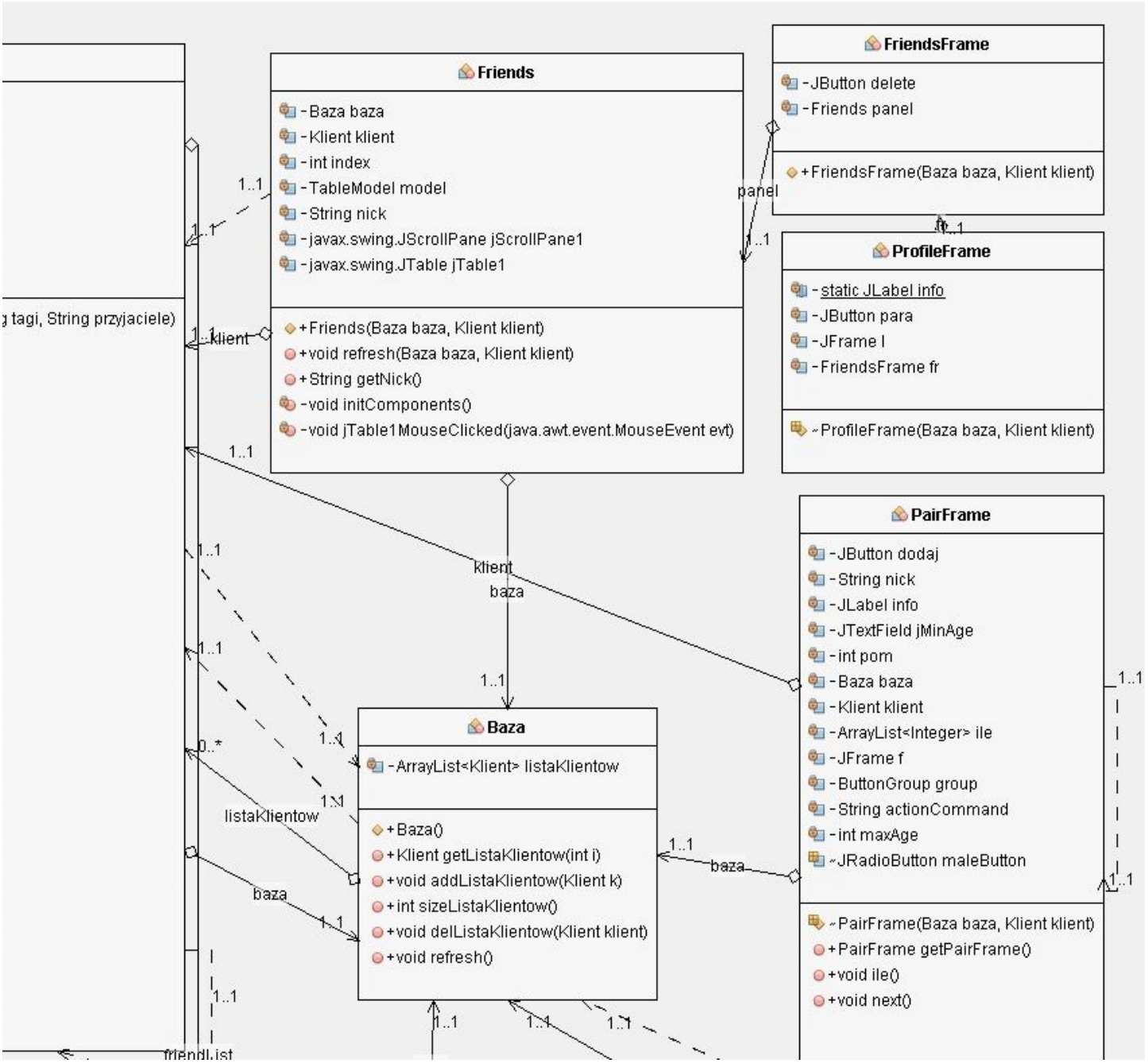
Niektóre utworzone obiekty są ważne dla działania całej aplikacji zatem zamknięcie ich powodowałoby niepotrzebne zamknięcie całej aplikacji.
Wyświetlone obiekty klasy MainFrame, NewFrame, OldFrame oraz ProfileFrame nie zamykają całej aplikacji od razu po naciśnięciu czerwonego przycisku EXIT w prawym górnym rogu lecz zostaje wyświetlony obiekt klasy Exit jako powiadomienie będące zawsze na wierzchu. Jeśli użytkownik kliknie przycisk “yes” aplikacja zostaje wyłączona, jeśli “no” powiadomienie znika. Inne wyświetlane obiekty również nie wyłączają aplikacji tylko zostają wyczyszczone za pomocą metody dispose().

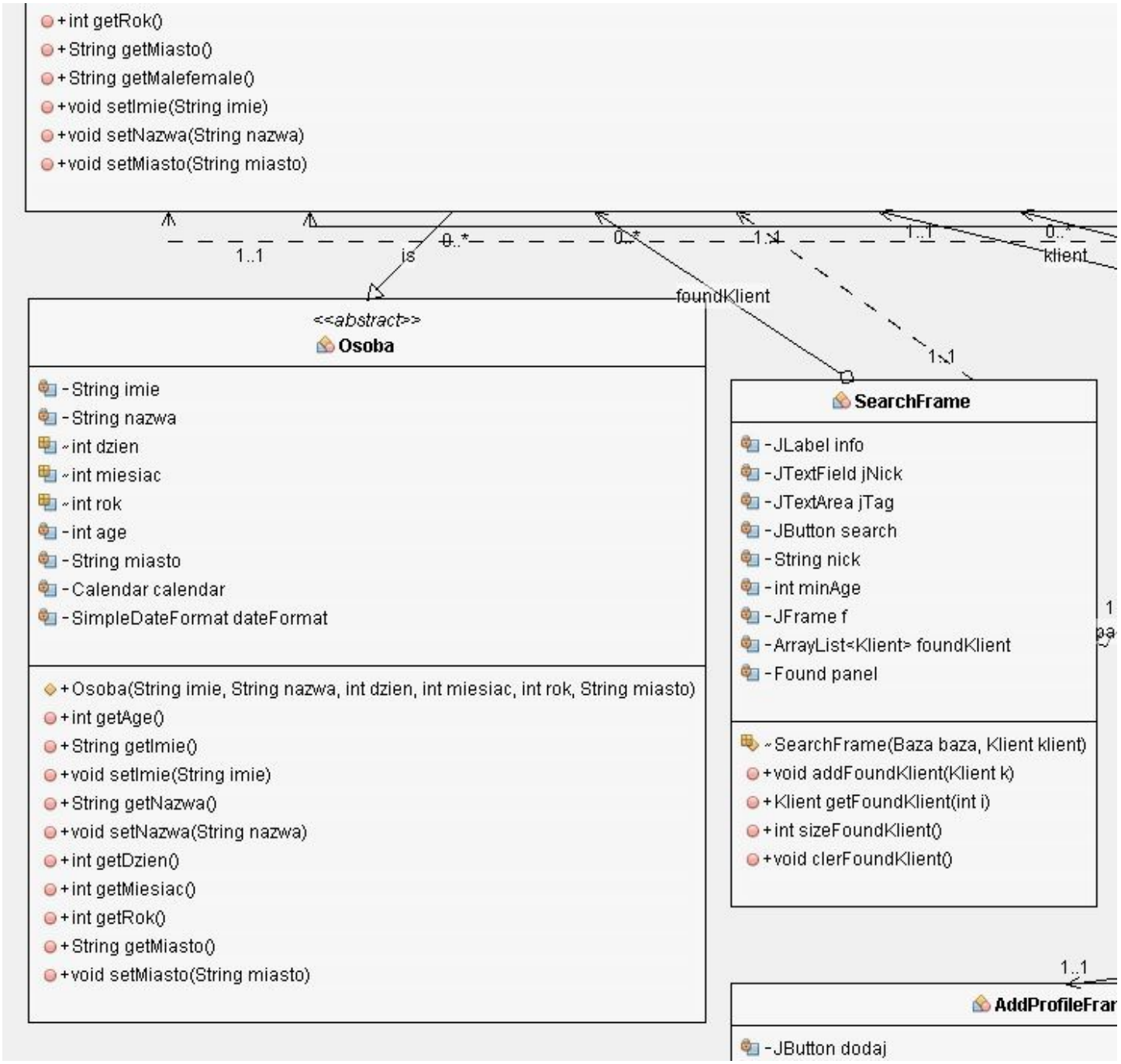
Podczas pierwszego uruchomienia programu bądź aktualizowania bazy danych z pliku program może nie znaleźć pliku z bazą dlatego gdy wystąpi wyjątek FileNotFoundException program tworzy wymagany plik i przy następnej próbie zaktualizownia baza już istnieje.

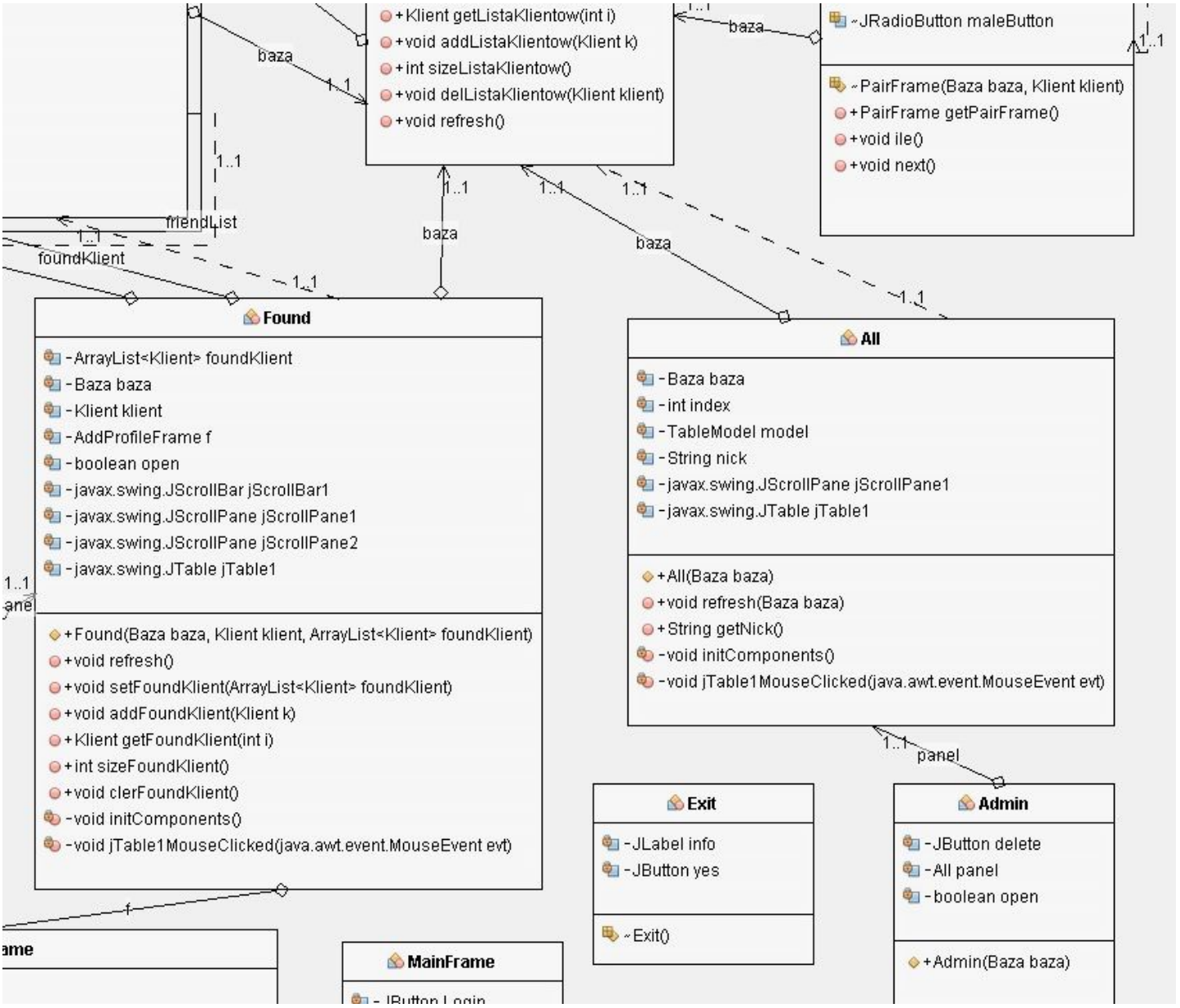
4. Diagramy UML

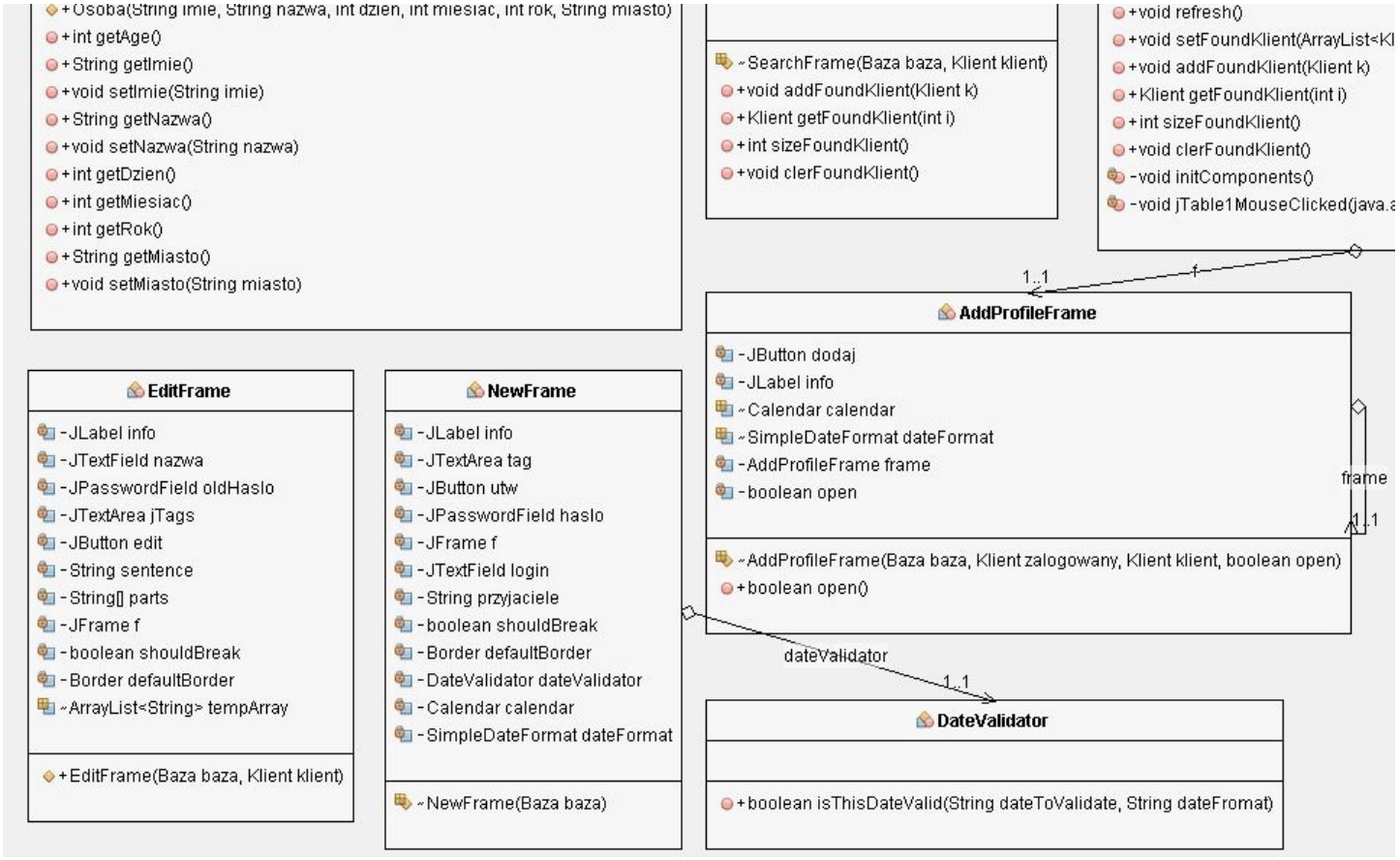


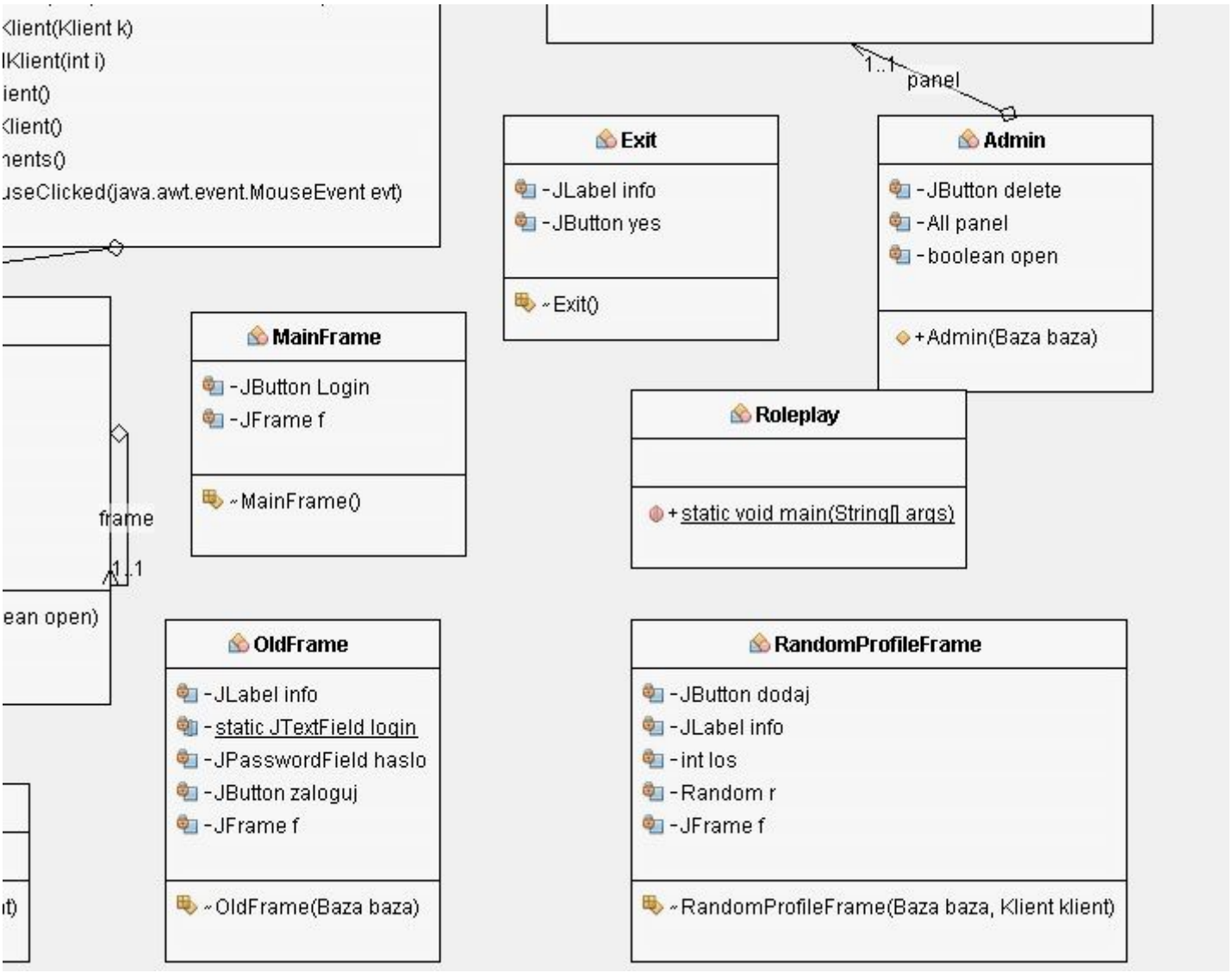

Klient
- String login
- String haslo
- String tagi
- String malefemale
- ArrayList<Klient> friendList
- Baza baza
- String przyjaciele
~ Klient(String login, String haslo, String imie, String nazwa, int dzien, int miesiac, int rok, String miasto, String malefemale, String tagi, String przyjaciele)
+void setPrzyjaciele()
+void addPrzyjaciele(Klient klient)
+void delPrzyjaciele(Klient klient)
+void refresh()
+String getPolish()
+int sizefriendList()
+Klient getPrzyjaciele(int i)
+String getPrzyjaciele()
+void addFriendList(Klient k)
+void clearFriendList()
+void setBaza(Baza baza)
+void addListaKlientow(Klient k)
+String getLogin()
+String getHaslo()
+String getTagi()
+void setTagi(String tagi)
+void setHaslo(String haslo)
+String getImie()
+String getNazwa()
+int getDzien()
+int getMiesiac()
+int getRok()
+String getMiasto()
+String getMalefemale()
+void setImie(String imie)
+void setNazwa(String nazwa)
+void setMiasto(String miasto)
1..1
klient
1..1
0..*
lista
1..1
friendList
1..1
is
0..*
0..*
1..1
1..1
0..*
klient
1..1
foundKlient


Friends
- Baza baza
- Klient klient
- int index
- TableModel model
- String nick
- javax.swing.JScrollPane jScrollPane1
- javax.swing.JTable jTable1
+ Friends(Baza baza, Klient klient)
+ void refresh(Baza baza, Klient klient)
+ String getNick()
- void initComponents()
- void jTable1MouseClicked(java.awt.event.MouseEvent evt)
FriendsFrame
- JButton delete
- Friends panel
+ FriendsFrame(Baza baza, Klient klient)
ProfileFrame
- static JLabel info
- JButton para
- JFrame l
- FriendsFrame fr
~ ProfileFrame(Baza baza, Klient klient)
PairFrame
- JButton dodaj
- String nick
- JLabel info
- JTextField jMinAge
- int pom
- Baza baza
- Klient klient
- ArrayList<Integer> ile
- JFrame f
- ButtonGroup group
- String actionCommand
- int maxAge
~ JRadioButton maleButton
~ PairFrame(Baza baza, Klient klient)
+ PairFrame getPairFrame()
+ void ile()
+ void next()
Baza
- ArrayList<Klient> listaKlientow
+ Baza()
+ Klient getListaKlientow(int i)
+ void addListaKlientow(Klient k)
+ int sizeListaKlientow()
+ void delListaKlientow(Klient klient)
+ void refresh()
panel
klient
baza
listaKlientow
friendList
1..1
0..*


+int getRok()
+String getMiasto()
+String getMalefemale()
+void setImie(String imie)
+void setNazwa(String nazwa)
+void setMiasto(String miasto)
1..1
is
0..*
0..*
1..1
1..1
0..*
klient
foundKlient
1..1
<<abstract>>
Osoba
-String imie
-String nazwa
~int dzien
~int miesiac
~int rok
-int age
-String miasto
-Calendar calendar
-SimpleDateFormat dateFormat
+Osoba(String imie, String nazwa, int dzien, int miesiac, int rok, String miasto)
+int getAge()
+String getImie()
+void setImie(String imie)
+String getNazwa()
+void setNazwa(String nazwa)
+int getDzien()
+int getMiesiac()
+int getRok()
+String getMiasto()
+void setMiasto(String miasto)
SearchFrame
-JLabel info
-JTextField jNick
-JTextArea jTag
-JButton search
-String nick
-int minAge
-JFrame f
-ArrayList<Klient> foundKlient
-Found panel
~SearchFrame(Baza baza, Klient klient)
+void addFoundKlient(Klient k)
+Klient getFoundKlient(int i)
+int sizeFoundKlient()
+void clerFoundKlient()
1..1
AddProfileFrar
-JButton dodaj


+Klient getListaKlientow(int i)
+void addListaKlientow(Klient k)
+int sizeListaKlientow()
+void delListaKlientow(Klient klient)
+void refresh()
~JRadioButton maleButton
~PairFrame(Baza baza, Klient klient)
+PairFrame getPairFrame()
+void ile()
+void next()
baza
1..1
friendList
foundKlient
Found
-ArrayList<Klient> foundKlient
-Baza baza
-Klient klient
-AddProfileFrame f
-boolean open
-javax.swing.JScrollBar jScrollBar1
-javax.swing.JScrollPane jScrollPane1
-javax.swing.JScrollPane jScrollPane2
-javax.swing.JTable jTable1
+Found(Baza baza, Klient klient, ArrayList<Klient> foundKlient)
+void refresh()
+void setFoundKlient(ArrayList<Klient> foundKlient)
+void addFoundKlient(Klient k)
+Klient getFoundKlient(int i)
+int sizeFoundKlient()
+void clerFoundKlient()
-void initComponents()
-void jTable1MouseClicked(java.awt.event.MouseEvent evt)
All
-Baza baza
-int index
-TableModel model
-String nick
-javax.swing.JScrollPane jScrollPane1
-javax.swing.JTable jTable1
+All(Baza baza)
+void refresh(Baza baza)
+String getNick()
-void initComponents()
-void jTable1MouseClicked(java.awt.event.MouseEvent evt)
panel
Exit
-JLabel info
-JButton yes
~Exit()
Admin
-JButton delete
-All panel
-boolean open
+Admin(Baza baza)
MainFrame
-JButton Login


+Osoba(String imie, String nazwa, int dzien, int miesiac, int rok, String miasto)
+int getAge()
+String getImie()
+void setImie(String imie)
+String getNazwa()
+void setNazwa(String nazwa)
+int getDzien()
+int getMiesiac()
+int getRok()
+String getMiasto()
+void setMiasto(String miasto)
~SearchFrame(Baza baza, Klient klient)
+void addFoundKlient(Klient k)
+Klient getFoundKlient(int i)
+int sizeFoundKlient()
+void clerFoundKlient()
+void refresh()
+void setFoundKlient(ArrayList<Kl
+void addFoundKlient(Klient k)
+Klient getFoundKlient(int i)
+int sizeFoundKlient()
+void clerFoundKlient()
-void initComponents()
-void jTable1MouseClicked(java.a
1..1
1
AddProfileFrame
-JButton dodaj
-JLabel info
~Calendar calendar
~SimpleDateFormat dateFormat
-AddProfileFrame frame
-boolean open
~AddProfileFrame(Baza baza, Klient zalogowany, Klient klient, boolean open)
+boolean open()
frame
1..1
EditFrame
-JLabel info
-JTextField nazwa
-JPasswordField oldHaslo
-JTextArea jTags
-JButton edit
-String sentence
-String[] parts
-JFrame f
-boolean shouldBreak
-Border defaultBorder
~ArrayList<String> tempArray
+EditFrame(Baza baza, Klient klient)
NewFrame
-JLabel info
-JTextArea tag
-JButton utw
-JPasswordField haslo
-JFrame f
-JTextField login
-String przyjaciele
-boolean shouldBreak
-Border defaultBorder
-DateValidator dateValidator
-Calendar calendar
-SimpleDateFormat dateFormat
~NewFrame(Baza baza)
dateValidator
1..1
DateValidator
+boolean isThisDateValid(String dateToValidate, String dateFromat)


lient(Klient k)
lKlient(int i)
ient()
lient()
nents()
useClicked(java.awt.event.MouseEvent evt)
1..1
panel
Exit
-JLabel info
-JButton yes
~Exit()
Admin
-JButton delete
-All panel
-boolean open
+Admin(Baza baza)
MainFrame
-JButton Login
-JFrame f
~MainFrame()
Roleplay
+static void main(String[] args)
frame
1..1
ean open)
OldFrame
-JLabel info
-static JTextField login
-JPasswordField haslo
-JButton zaloguj
-JFrame f
~OldFrame(Baza baza)
RandomProfileFrame
-JButton dodaj
-JLabel info
-int los
-Random r
-JFrame f
~RandomProfileFrame(Baza baza, Klient klient)