Criterion C: Development

UML diagram

The diagram below gives an overview of dependencies in the program I created for Ms XX.

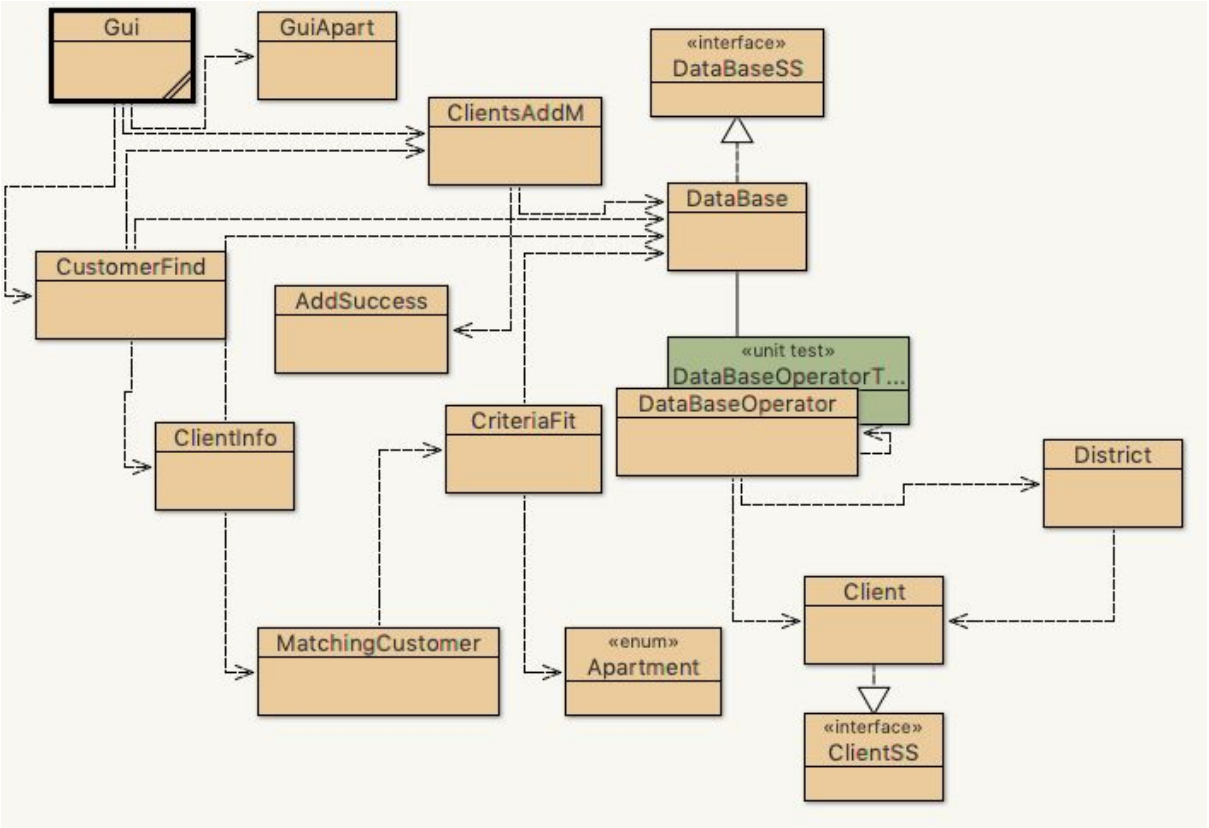


Fig. 1.Print screen of classes from BlueJ

Techniques used in the solution:

- Graphic User Interface (graphical libraries, ActionListeners)
- Converting passed values into parameters for Client (try & catch, exception handling)
- Parallel arrays
- Enumerated types
- Linear search for the max value
- ArrayLists
- Polymorphism
- Encapsulation
- Inheritance
- Singleton class
- Hierarchical composite data structure
- Reading and writing into a txt file
- Unit testing
- Interface

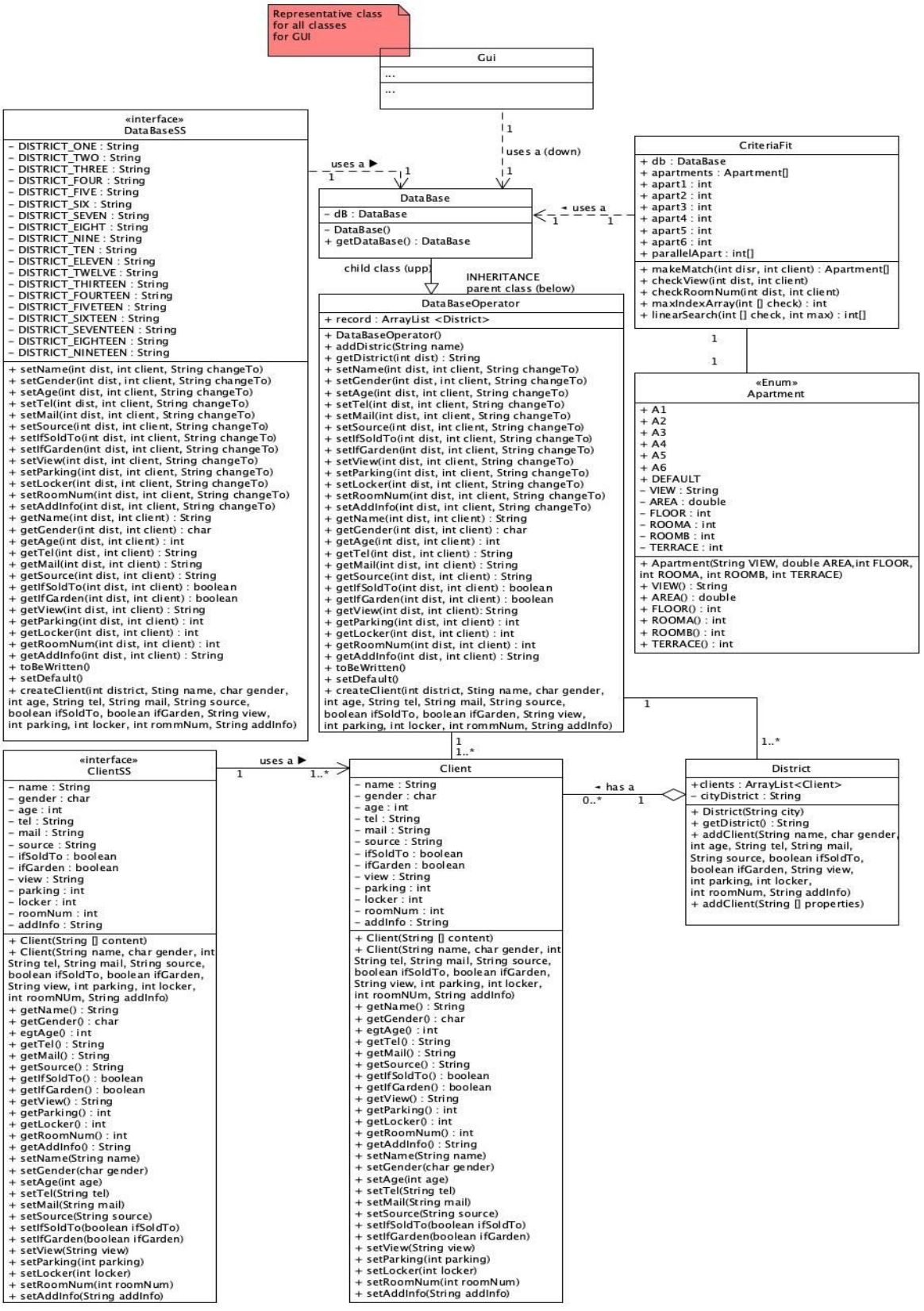


Fig. 2. UML diagram for my program (with a representative class for all GUI classes and omitted unit testing class)

In figure 2, the dependencies between objects are displayed:

- association (CriteriaFit utilizes an Apartment, DataBaseOperator utilizes one or more clients and one or more districts)
- dependency, objects are dependent on one another (Gui uses a DataBase, DataBaseSS uses a DataBase, CriteriaFit uses a DataBase)
- inheritance, child classes inherit methods from parent class (DataBaseOperator is a DataBase)
- aggregation, one object belongs to the other (Client has a District)

Program from the perspective of the client.

The client runs the program by right-clicking on the Gui class and pressing enter. The welcome window pops up:

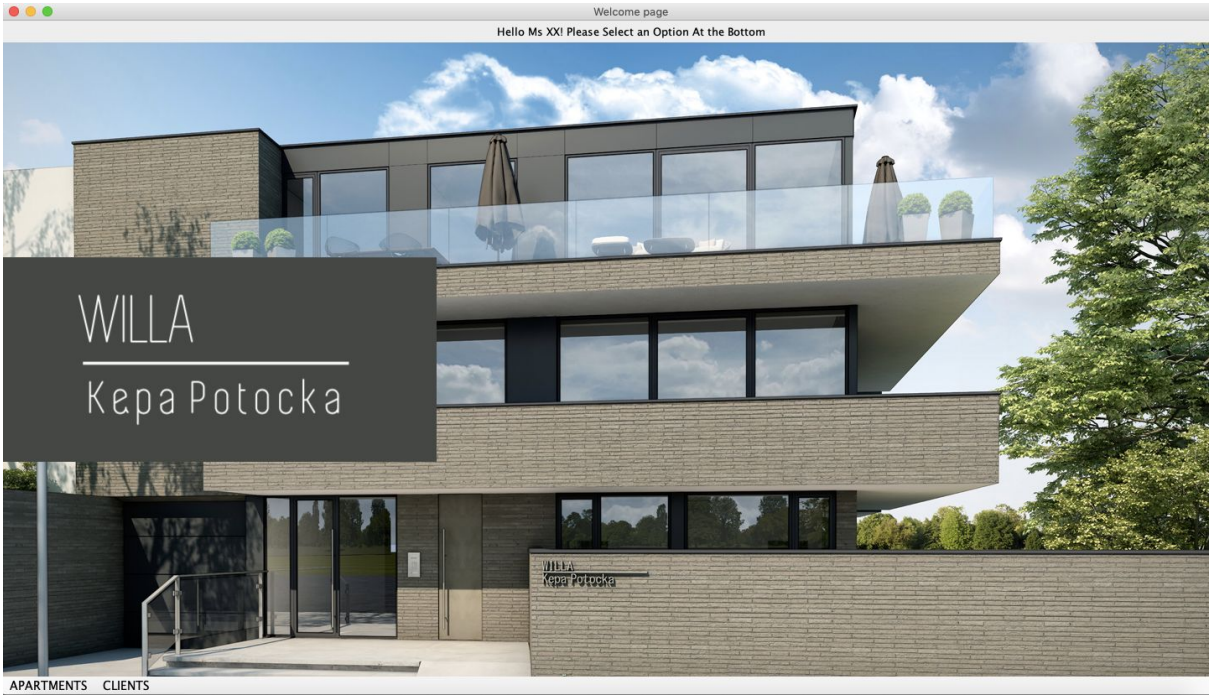


Fig. 3. Welcome window

For creating the GUI I imported the following **libraries**:

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
```

Fig. 4. importing libraries for Gui class

It allowed me to set up actions after a selected menu item. Setting up the window, creating JMenu in a panel and them JMenuItems for each apartment, instantiating classes GuiApart, ClientsAddM, CustomerFind.

```
public static void Gui(String[] args)
{

    GuiApart apart = new GuiApart();
    ClientsAddM add = new ClientsAddM();
    CustomerFind find = new CustomerFind();

    //Creating the Frame
    JFrame frame = new JFrame("Welcome page");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(1500, 950);

    //Creating the MenuBar and adding components
    JMenuBar optionBar = new JMenuBar();
    JMenu mApartments = new JMenu("APARTMENTS");
    JMenu mClients = new JMenu("CLIENTS");
    optionBar.add(mApartments);
    optionBar.add(mClients);

    JMenuItem mA1 = new JMenuItem("Apartment 1");
    JMenuItem mA2 = new JMenuItem("Apartment 2");
    JMenuItem mA3 = new JMenuItem("Apartment 3");
    JMenuItem mA4 = new JMenuItem("Apartment 4");
    JMenuItem mA5 = new JMenuItem("Apartment 5");
    JMenuItem mA6 = new JMenuItem("Apartment 6");
```

Fig. 5. creating menu and menu items in Gui

```
    //adding components
    mApartments.add(mA1);
    mApartments.add(mA2);
    mApartments.add(mA3);
    mApartments.add(mA4);
    mApartments.add(mA5);
    mApartments.add(mA6);

    //creating components
    JMenuItem mClient = new JMenuItem("Clients");
    JMenuItem mClientAdd = new JMenuItem("Add client");
```

Fig. 6. Adding the created components into the menu

```
    //adding components into menu
    mClients.add(mClient);
    mClients.add(mClientAdd);

    //Creating the panel at bottom and adding components
    JPanel panel = new JPanel();
    JLabel label = new JLabel("Hello Ms XX! Please Select an Option At the Bottom");

    panel.add(label);


    //creating JLabels for images
    ImageIcon willa = new ImageIcon("willa.png");
    JLabel labelWilla = new JLabel (willa);
    ImageIcon front = new ImageIcon("elewacja.png");
    JLabel labelFront = new JLabel(front);


    //Adding Components to the frame.
    frame.getContentPane().add(BorderLayout.NORTH, panel);
    frame.getContentPane().add(BorderLayout.SOUTH, optionBar);
    frame.getContentPane().add(BorderLayout.WEST, labelWilla);
    frame.getContentPane().add(BorderLayout.EAST, labelFront);
    frame.setVisible(true);
```

Fig. 7. Setting the layout, adding components into the ContentPane

When the client selects “APARTMENTS” from the menu at the bottom a list shows up

Fig. 8. List of Apartments

For the GUI menu items, I utilised methods from ActionEvent and ActionListener classes. (ActionEvent (Java Platform SE 7), n.d.). For each menu item, there is an ActionListener to see if the item was selected.

```
//if menu item mA1 is selected then call method guiA1
mA1.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent arg0)
    {
        apart.guiA1();
    }
});
```

Fig. 9. Action listener for one of the menu items for Apartments

When selecting Apartment 1 from the list, new window with the layout of the apartment and its properties pops up.

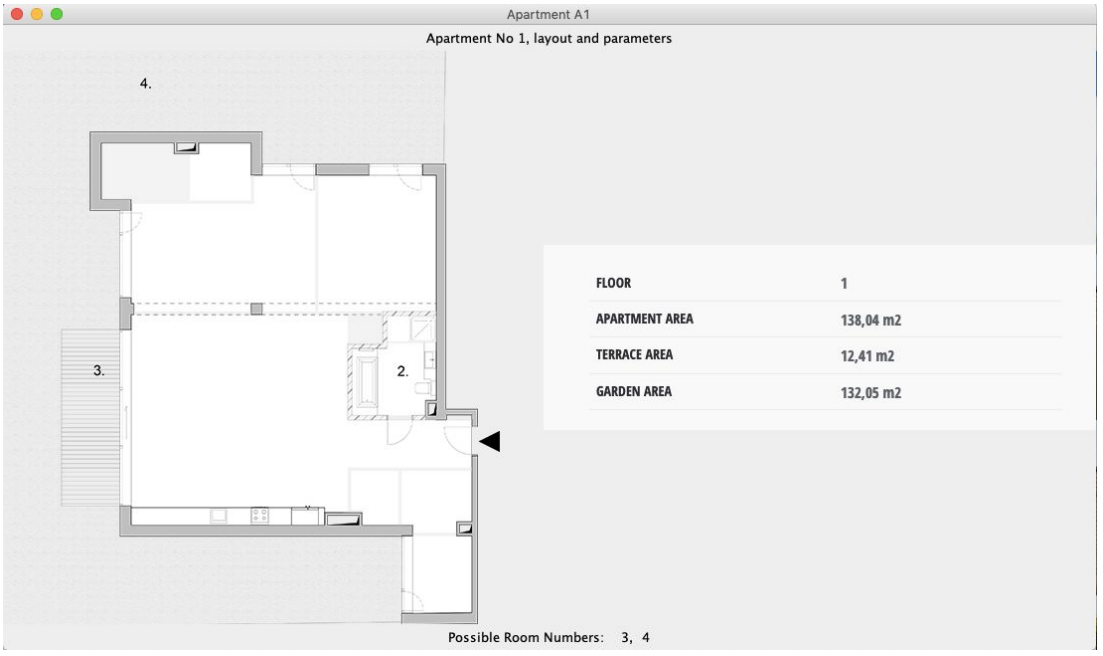


Fig. 10. Window for an apartment

```
//setting up a frame for the GUI
JFrame frame = new JFrame("Apartment A1");

//dispose will close only this frame
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
//adjust size
frame.setSize(1100, 650);

//construct components
JPanel panel = new JPanel();
JLabel label = new JLabel("Apartment No 1, layout and parameters");
panel.add(label);

ImageIcon layout = new ImageIcon("A1.png");
JLabel labelLayout = new JLabel (layout);

ImageIcon values = new ImageIcon("a1Values.png");
JLabel labelValues = new JLabel(values);

//creating panel1
JPanel panel1 = new JPanel();
//construct componetnts for panel1
JLabel rNum = new JLabel("Possible Room Numbers:   ");
JLabel roomA = new JLabel("3, ");
JLabel roomB = new JLabel("4");

//add components to the panel1
panel1.add(rNum);
panel1.add(roomA);
panel1.add(roomB);
```

Fig. 11. Disposing of the window of an Apartment

The code above shows how I created the windows for all 6 of the apartments, setting up a frame. JFrame.DISPOSE_ON_CLOSE allows for the program to run when this window is closed, the client will still have the welcome window open and can use the program without rerunning.

The list after selecting “CLIENTS” from the menu:

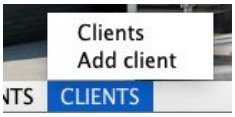


Fig. 12. List for CLIENTS menu item

The windows mentioned earlier were designed by me using a BorderLayout. For the following windows, I used GuiGenie generator (Awad, n.d.). This window pops up when Ms XX selects “Add client” . It uses the JTextFields, JComboBoxes, JCheckBoxes for providing the information on the client. An action listener is used for button OK.

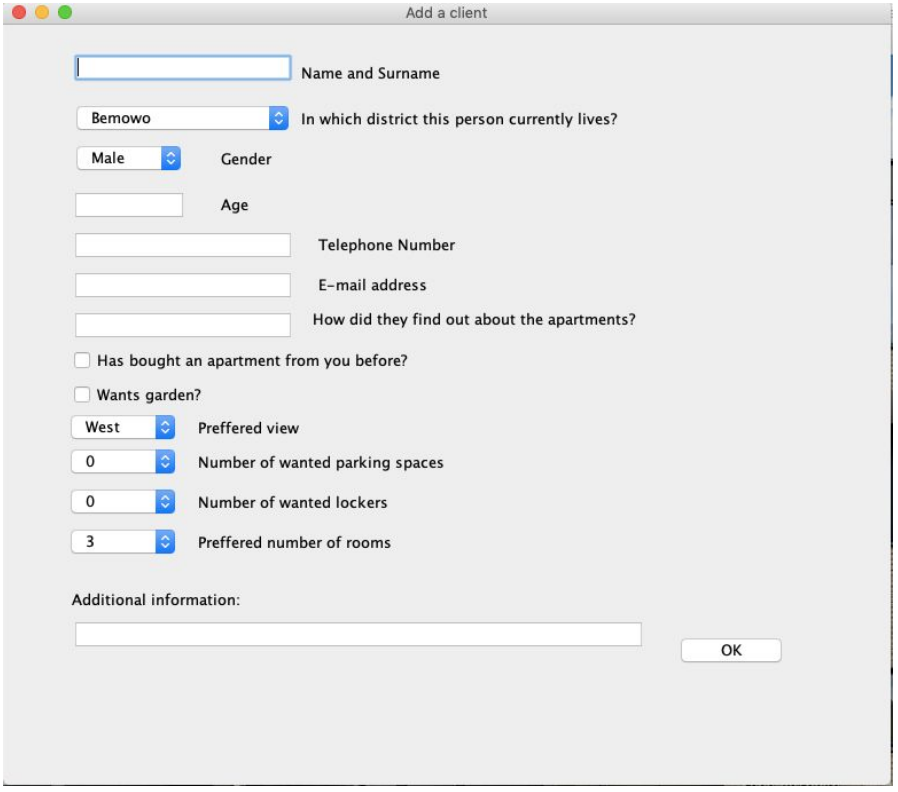


Fig. 13. Window for adding a client

Converting the provided values into parameters for creating a client.

```
name = fieldName.getText();

//converting selected item to String
String rStr = boxRooms.getSelectedItem().toString();
//String to int
roomNum = Integer.parseInt(rStr);

//getting first char of a String
char temp =  boxView.getSelectedItem().toString().charAt(0);
//converting the char to String
view = Character.toString(temp);

String dist = String.valueOf(listDistr.getSelectedItem());
district = getDistNum(dist);
```

Fig. 14. Example of converting provided values into parameters for object Client

A method getDistNum(dist) converts the district selected from the list into its corresponding index (code only for the first 7 districts displayed). It checks each case of the switch statement for integer dist.

```
public int getDistNum(String dist)
{
    int value = -1;
    switch(dist)
    {
        case "Bemowo":
        value = 0;
        break;

        case "Bialoleka":
        value = 1;
        break;

        case "Bielany":
        value = 2;
        break;

        case "Mokotow":
        value = 3;
        break;

        case "Ochota":
        value = 4;
        break;

        case "Praga-Południe":
        value = 5;
        break;

        case "Praga-Polnoc":
        value = 6;
        break;

        case "Rembertow":
        value = 7;
```

Fig. 15. Switch case statement for converting the selected district into corresponding integer

When the client selects from the welcome window “Clients”, she can find a client by selecting a district from the list

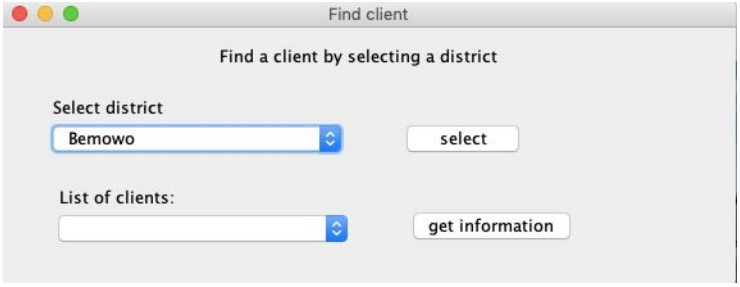


Fig. 16. Window for finding a client

After clicking on select, the list of client is cleared and then filled with clients for the district. This allows Ms XX to change the dirstricts without rerunning the window.

```
//remove items from boxNames
boxNames.removeAllItems();

//converting selected item String into int
String dist = String.valueOf(boxDistricts.getSelectedItem());
district = client.getDistNum(dist);

//looping through the size of the record for this district
for(int index = 0; index < db.record.get(district).clients.size();
    index = index + 1)
{
    //adding all names in this district into the boxNames
    boxNames.addItem(db.getName(district, index));
}
```

Fig. 17. clearing the list with removeAllItems()

After clicking “get information” window with the information on client appears.

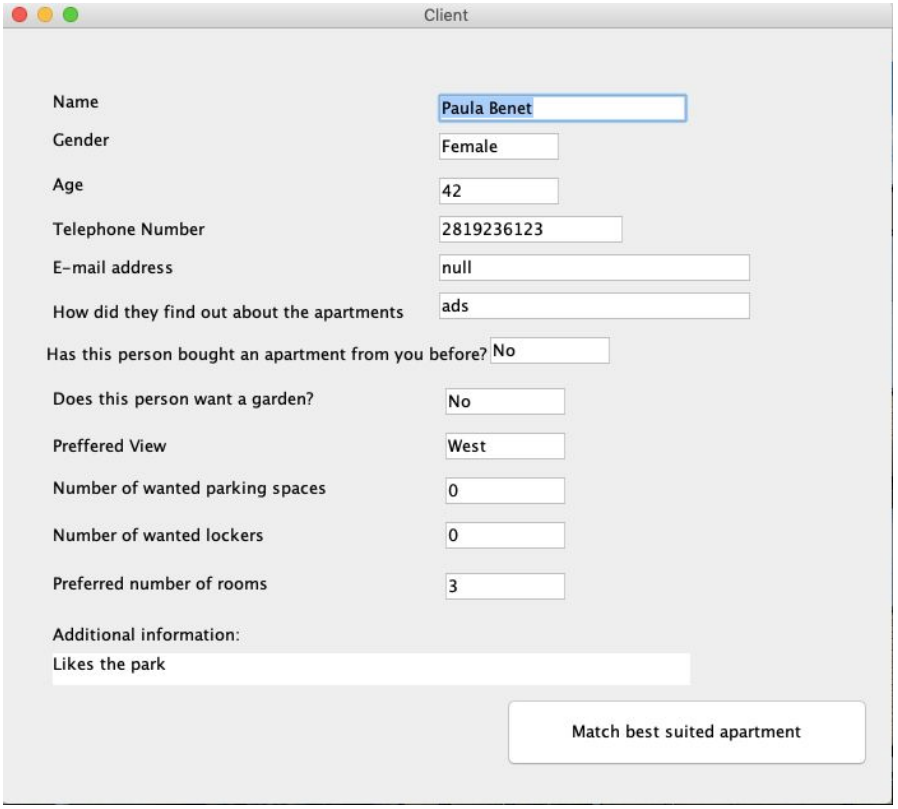


Fig. 18. Window for displaying information on a selected client

It calls the value for the boolean IfGarden for this client and outputs an appropriate String in an appropriate JTextField

```
public String getGarden(int dist, int client)
{
    boolean wantsGarden = db.getIfGarden(dist, client);

    if(wantsGarden) {return "Yes";}
    else {return "No";}
}
```

Fig. 19. Converting the value of boolean IfGarden in Client to appropriate String

For matching after pressing the button “Match best suited apartment”

Fig. 20. Window for matched apartment(s)

The algorithm for matching an apartment utilizes parallel arrays as one stores enumerated types of Apartment and the other stores corresponding integer values for matching. They are used in methods described later.

```
//array of enum apartments
Apartment [] apartments = {Apartment.A1, Apartment.A2, Apartment.A3,
                           Apartment.A4, Apartment.A5, Apartment.A6};


//array of values for matching each of the aapartment to client's needs
int[] parallelApart = {apart1, apart2, apart3, apart4, apart5, apart6};
```

Fig. 21. Use of paralle arrays

Used enumerated types :

```
A1 ("A1","W", 138.04 ,1, 4, 3, 1),
A2 ("A2", "E",75.72, 1, 3, 3, 1 ),
A3 ("A3", "W",144.77, 2, 5, 4, 2),
A4 ("A4", "E", 115.69, 2, 4, 3, 1),
A5 ("A5", "W", 144.77, 3, 5, 4, 2),
A6 ("A6", "E", 104.26, 3, 4, 3, 1),
DEFAULT ("","", 0.0, 0,0,0,0);
```

Fig. 22. Enumerated types for Apartment

To represent a group of constants of the properties apartments as they won’t change.

```
/*
 * enum constructor
 */
Apartment(String NAME, String VIEW, double AREA, int FLOOR, int ROOMA,
            int ROOMB, int TERRACE)
{
    this.NAME = NAME;
    this.VIEW = VIEW;
    this.AREA = AREA;
    this.FLOOR = FLOOR;
    this.ROOMA = ROOMA;
    this.ROOMB = ROOMB;
    this.TERRACE = TERRACE;
}

//set of accessors
public String NAME(){return NAME;}
public String VIEW() {return VIEW;}
public double AREA(){return AREA;}
public int FLOOR(){return FLOOR;}
public int ROOMA(){return ROOMA;}
public int ROOMB(){return ROOMB;}
public int TERRACE(){return TERRACE;}
```

Fig. 23 Constructor and accessor for enumerated types

Method makeMatch returns an array of the apartments with the highest match. If the customer wants a garden the value for match in parallelApart for the first and second apartment is increased.

```
if(db.getIfGarden(dist, client) == true)
{
    //adding one point to apart1 and apart2 as only these have a gardern
    apart1 = apart1 + 1;
    apart2 = apart2 + 1;

    //changing the assigned values of matching for the apartment in
    //parallel array of parallelApart
    parallelApart[0] = apart1;
    parallelApart[1] = apart2;
}

parallelApart = checkView(dist, client);
parallelApart = checkRoomNum(dist, client);

//array of indexes at which the max value is stored
int [] equalSuit = linearSearch(parallelApart, maxValueArray(parallelApart));

//array of Apartment
Apartment [] bestSuit = new Apartment [equalSuit.length];

//getting the corresponding Apartment enum data type for equalSuit
for(int index = 0; index < equalSuit.length; index = index + 1)
{
    bestSuit[index] = apartments[equalSuit[index]];
}


return bestSuit;
```

Fig. 24. Method makeMatch(int dist, int client)

Methods checkView() and checkRoomNum() increase the values in parallelApart for the apartments which have the properties wanted by the client.

```
/*
 * method CHECKVIEW(CLIENT)
 *  loop INDEX from 0 to APARTMENTS.length
 *     if APARTMENTS[INDEX].VIEW() = CLIENT.getView() then
 *           PARALLELAPART[INDEX] = PARALLELAPARTINDEX + 1
 *     end if
 *   end loop
 * end method
 */
public int[] checkView(int dist, int client)
{
    //getting view from db (dB instance of DataBase)
    String clientView = db.getView(dist, client);
    String apartView = "";

    //looping through the apartments array
    for (int index = 0; index < apartments.length; index = index +1)
    {
        //setting apartView to VIEW in an Apartment
        apartView = apartments[index].VIEW();

        if (clientView.equals(apartView))
        {
            parallelApart[index] = parallelApart[index] + 1;
        }
    }

    return parallelApart;
}
```

Fig. 25. Method chekcView(int dist, int client)

```
/* method CHECKROOMNUM(CLIENT)
 *  loop INDEX from 0 to APARTMENTS.length
 *     if APARTMENTS[INDEX].ROOMA = CLIENT.getRoomNum() OR
 *        APARTMENTS[INDEX].ROOMB = CLIENT.getRoomNum() then
 *        PARALLELAPART[INDEX] = PARALLELAPART[INDEX] + 1
 *     end if
 *  end loop
 * end method
 */
public int[] checkRoomNum(int dist, int client)
{
    //looping through apartments array
    for (int index = 0; index < apartments.length; index = index + 1)
    {
        //checking if either ROOMA or ROOMB for the Apartment is the same
        //as the RoomNum wanted by the client
        if(db.getRoomNum(dist, client) == apartments[index].ROOMA() ||
        db.getRoomNum(dist, client) == apartments[index].ROOMB())
        {
            parallelApart[index] = parallelApart[index] + 1;
        }
    }

    return parallelApart;
}
```

Fig. 26. Method checkRoomNum(int dist, int client)

```
/*
 * method MAXVALUEARRAY(PARALLELAPART)
 *  MAX = PARALLELAPART[0]
 *
 *  loop INDEX from 1 to PARALLELAPART.length
 *     if PARALLELAPART > MAX then
 *          MAX = PARALLELAPART[INDEX]
 *      end if
 *  end loop
 *
 *  return MAX
 * end method
 */
public int maxValueArray(int [] check)
{
    //initial max to 0
    int max = check[0];

    //looping though the array check
    for(int index = 1; index < check.length; index = index + 1)
    {
        //if value stored in the check array at index is greater than max
        //max becomes the value
        if(check[index] > max)
        {
            max = check[index];
        }
    }
    return max;

}
```

Fig. 27. Method maxValueArray(int [] check)

Method maxValueArray searches linearly for the highest value in the array, which then is passed as an argument into linearSearch which returns an array of indexes at which the max is stored

```
public int [] linearSearch(int [] check, int max)
{
     //ArrayList of intergers to store indexes at which the maximum is
     List<Integer> indexesAt = new ArrayList<Integer>();

    //adding the indexes to the ArrayList
    for(int index = 0; index < check.length; index = index + 1)
    {
        //when the value stored at the index is the maximum, the index is added
        //to the ArrayList
        if (max == check[index])
        {
            indexesAt.add(index);
        }
    }

    //creating an array of size of the ArrayList
    int [] result = new int [indexesAt.size()];

    //storing the indexes in array
    for(int index = 0; index < indexesAt.size(); index = index + 1)
    {
        result[index] = indexesAt.get(index);
    }

    return result;
}
```

Fig. 28. Use of ArrayList for storing unknown number of values

ArrayList indexesAt allows for the dynamic adding of the indexes, without having to assign the size of it. First loop adds indexes to the ArrayList, then an array of its size is created and filled with the stored indexes in the second loop.

Methods for data base

I started with creating a hierarchical composite data structure. Firstly, in the Client class, I implemented encapsulation of attributes in creating the Client object. The private data can only be accessed through public accessor methods.

```
import java.util.*;
import java.util.ArrayList;
import java.util.List;

public class Client implements ClientSS
{
    private String name;
    private char gender;
    private int age;
    private String tel;
    private String mail;
    private String source;
    private boolean ifSoldTo;
    private boolean ifGarden;
    private String view;
    private int parking;
    private int locker;
    private int roomNum;
    private String addInfo;
```

Fig. 29. Declaration of private variables

For the creation of Client objects I utilised polymorphism, with methods for creating Client taking either an array of String or separate parameters.

```
/*
 * two methods Client take different parameters,
 * compile time polymorphism
 */
public Client(String[] content)
{
    name = content[0];

    String temp;
    temp = content[1];
    gender = temp.charAt(0);

    age = Integer.parseInt(content[2]);
    tel = content[3];
    mail = content[4];
    source = content[5];
    ifSoldTo = Boolean.parseBoolean(content[6]);
    ifGarden = Boolean.parseBoolean(content[7]);
    view = content[8];
    parking = Integer.parseInt(content[9]);
    locker = Integer.parseInt(content[10]);
    roomNum = Integer.parseInt(content[11]);
    addInfo = content[12];
}
```

Fig. 30. Polymorphism, Client method for array of String

```
/*
 * different parameters for this method than the previous one.
 * Depending on the parameters provided when calling Client method
 * different method will be called
 *
 */
public Client(String name, char gender, int age, String tel, String mail,
            String source, boolean ifSoldTo, boolean ifGarden, String view,
            int parking, int locker, int roomNum, String addInfo)
{
    this.name = name;
    this.gender = gender;
    this.age = age;
    this.tel = tel;
    this.mail = mail;
    this.source = source;
    this.ifSoldTo = ifSoldTo;
    this.ifGarden = ifGarden;
    this.view = view;
    this.parking = parking;
    this.locker = locker;
    this.roomNum = roomNum;
    this.addInfo = addInfo;
}
```

Fig. 31. Polymorphism, Client method for separate parameters

Then in District class, I created an ArrayList for storing Client objects names clients.

```
public class District
{
    //creaing an ArrayList of Client class
    List<Client> clients = new ArrayList<>();
    private String cityDistrict;

    //instantiating District class constructor
    public District(String city){this.cityDistrict = cityDistrict;}

    //accessor for getting the String of cityDistrict
    public String getDistrict(){return cityDistrict;}
```

Fig. 32. ArrayList of class Client

Later in DataBaseOperator class, I set up an ArrayList for storing District objects. This hierarchical composite data structure allows for a record like data structure. Set up in the DataBaseOperator class record is an ArrayList of District class object, which has an ArrayList of Client class objects.

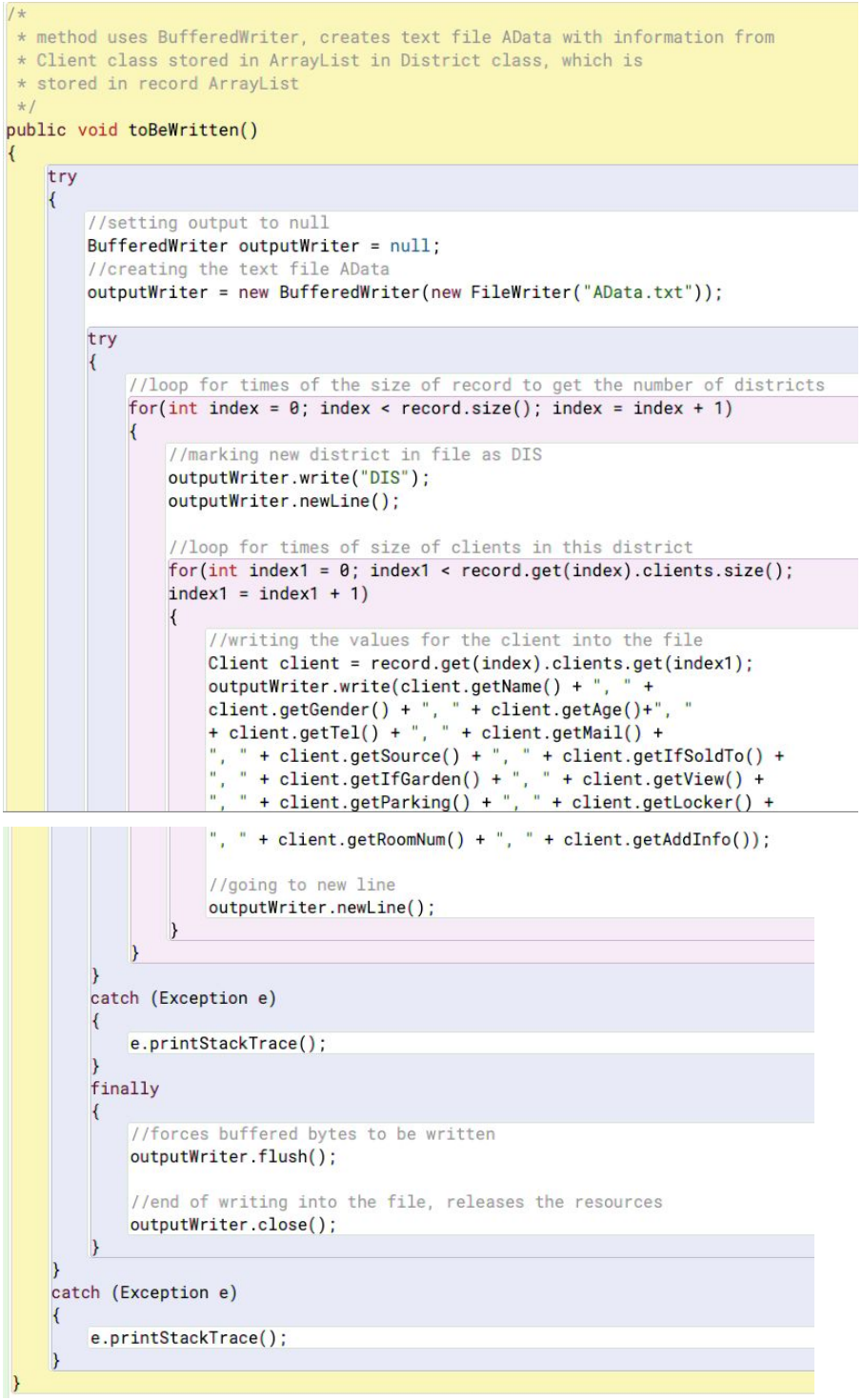
```
public class DataBaseOperator
{
    //record is an ArrayList of District class
    List<District> record = new ArrayList<>();
```

Fig. 33. ArrayList of class District

Using this record like data structure I moved onto setting up reading and writing into a text file using methods from FileReader and BufferedWriter classes.

The method toBeWritten() writes into the AData.txt file, utilising exception handling for catching any errors. The first for loop goes through all objects of District stored in the record, writes “DIS” for every new district and enters the nested for loop. This one iterates for as many times as there are clients stored in this specific district. It then writes into the file the client with all its properties. After

completion of the processes without errors, it goes into finally bracket where everything that might be buffering is written and the resources are freed.



```
/*
 * method uses BufferedWriter, creates text file AData with information from
 * Client class stored in ArrayList in District class, which is
 * stored in record ArrayList
 */
public void toBeWritten()
{
    try
    {
        //setting output to null
        BufferedWriter outputWriter = null;
        //creating the text file AData
        outputWriter = new BufferedWriter(new FileWriter("AData.txt"));

        try
        {
            //loop for times of the size of record to get the number of districts
            for(int index = 0; index < record.size(); index = index + 1)
            {
                //marking new district in file as DIS
                outputWriter.write("DIS");
                outputWriter.newLine();

                //loop for times of size of clients in this district
                for(int index1 = 0; index1 < record.get(index).clients.size();
                index1 = index1 + 1)
                {
                    //writing the values for the client into the file
                    Client client = record.get(index).clients.get(index1);
                    outputWriter.write(client.getName() + ", " +
                    client.getGender() + ", " + client.getAge()+", "
                    + client.getTel() + ", " + client.getMail() +
                    ", " + client.getSource() + ", " + client.getIfSoldTo() +
                    ", " + client.getIfGarden() + ", " + client.getView() +
                    ", " + client.getParking() + ", " + client.getLocker() +
                    ", " + client.getRoomNum() + ", " + client.getAddInfo());

                    //going to new line
                    outputWriter.newLine();
                }
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        finally
        {
            //forces buffered bytes to be written
            outputWriter.flush();

            //end of writing into the file, releases the resources
            outputWriter.close();
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
```

Fig. 34. Writing into the file

Next method textToRead() reads the text from AData file, reading one line at a time. With each “DIS” in the file, the program assigns the client to the next district and increases the integer district. If the line read is not “DIS” then it adds to the client to the district-1.

```java
/*
 * method reads the AData textFile into the record ArrayList
 */
public void textToRead()
{
    try
    {
        int dist = 0;
        int client = 0;
        FileReader data = new FileReader("AData.txt");
        BufferedReader reader = new BufferedReader(data);
        while(reader.ready())
        {
            //reading one line at a time
            String justRead = reader.readLine();

            switch(justRead)
            {
                //if the line is only DIS then it means it is a new district
                case "DIS" :
                //setting number of clients in a district to be 0
                client = 0;
                addDistrict(justRead);

                //adding client read on the next line
                //distinguishing the parameters by ", "
                record.get(dist).addClient((reader.readLine()).split(", "));

                //increasing the district number, next district
                dist = dist + 1;
                break;

                //default case, justRead is not "DIS"
                default:
                record.get(dist-1).addClient(justRead.split(", "));

            }
        }
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

Fig. 35. Reading the file

For checking whether the methods work correctly I did a JUnitTest for getting/setting name, age gender (only testgetName() displayed below). I learned it on stackoverflow website (Krejcha, 2012).

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.io.*;
import java.util.*;

/**
 * The test class DataBaseOperatorTest.
 *
 */
public class DataBaseOperatorTest
{
    @Test
    public void testGetName()
    {
        DataBaseOperator data1 = new DataBaseOperator();
        data1.textToRead();
        assertEquals("Agnieszka Kowalska", data1.getName(0, 0));
    }
```

Fig. 36. Example of JUnitTest

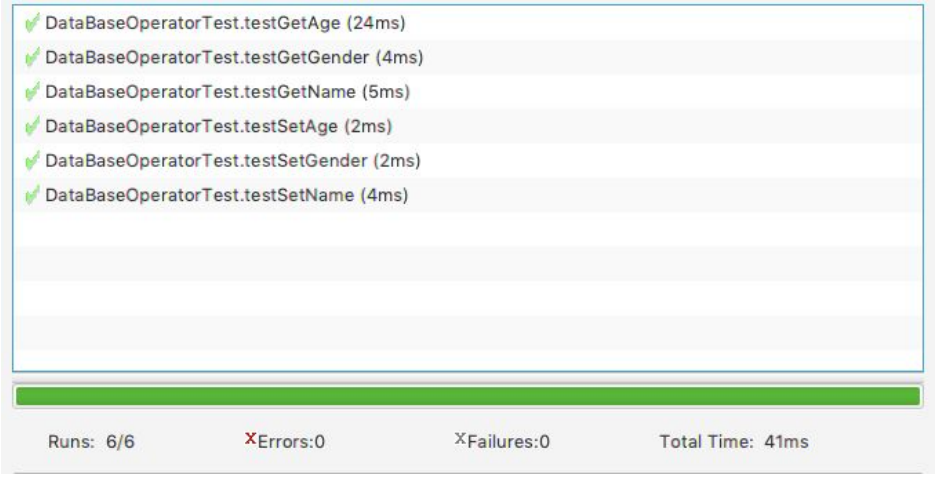


Fig. 37. Results of the test

After testing the methods, all with positive outcomes, it reassured me that the methods were working correctly.

For the DataBase class called from other classes, the one instance dB is returned. This class is a singleton, there is only one instance of it at a time. (Rayapati, n.d.).

```
/**
 *  singleton DataBase class, is only once instanciated
 *  only one dB object is returned to callers
 */
public class DataBase extends DataBaseOperator implements DataBaseSS
{
    //static variabe dB of type DataBase
    private static final DataBase dB = new DataBase();

    private DataBase(){textToRead();}

    //returns the static instance created in the Data section above
    public static DataBase getDataBase(){return dB;}

}
```

Fig. 38. Singleton of DataBase

A singleton of DataBase class allowed me to operate on one file (AData), one object of DataBase created and throughout the program, only this instance is used

I added interfaces for the classes DataBase and Client, a description of the methods included in these classes. (Java - Interfaces - Tutorialspoint, n.d.). It allowed me to have a quick overview of the methods. The interface and all its methods are abstract, with the latter being implicitly public.

```
public interface ClientSS
{
    String name = "client's name";
    char gender = 'n';
    int age = 0;
    String tel = "0";
    String mail = "client's mail";
    String source = "";

    boolean ifSoldTo = false;
    boolean ifGarden = false;

    String view = "";
    int parking = 0;
    int locker = 0;
    int roomNum = 0;

    //accessors
    public String getName();
    public char getGender();
    public int getAge();
    public String getTel();
    public String getMail();
    public String getSource();
    public boolean getIfSoldTo();
    public boolean getIfGarden();
    public String getView();
    public int getParking();
    public int getLocker();
    public int getRoomNum();
```

Fig. 39. Interface for Client class

Method Summary

All Methods | **Instance Methods** | **Abstract Methods**

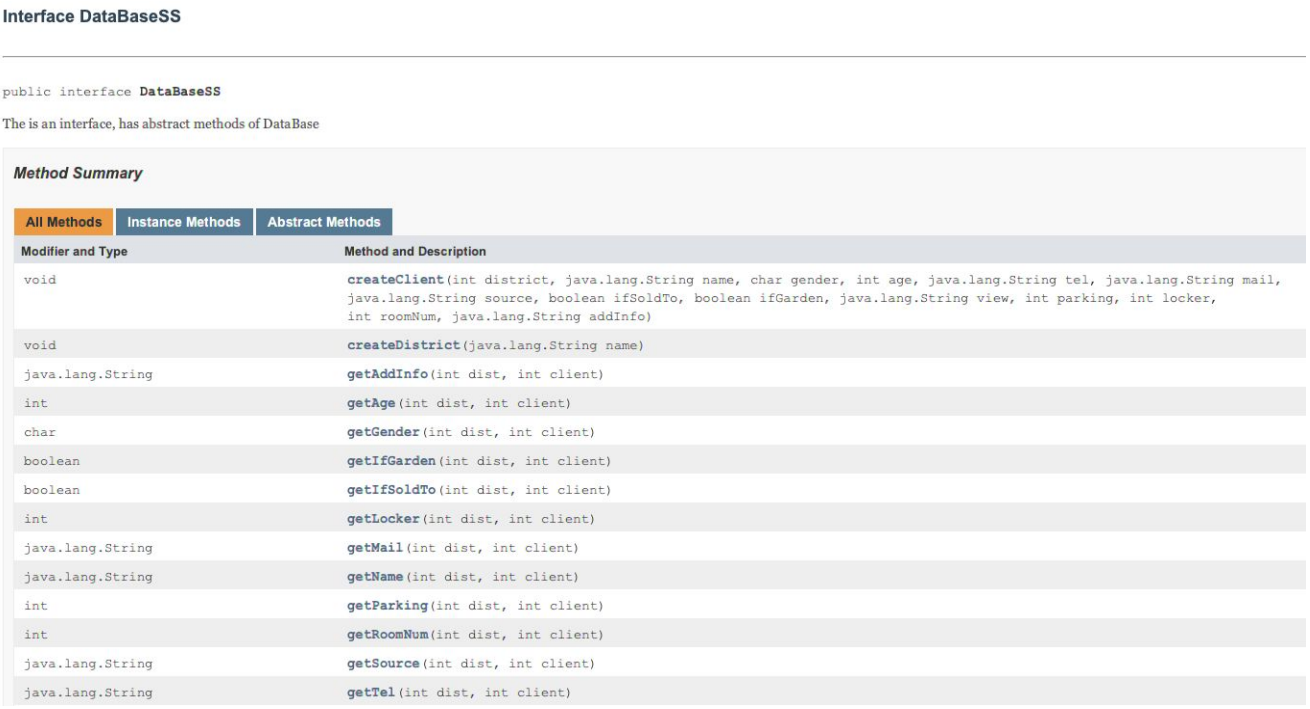
Modifier and Type	Method and Description
int	**getAge**()
char	**getGender**()
boolean	**getIfGarden**()
boolean	**getIfSoldTo**()
int	**getLocker**()
java.lang.String	**getMail**()
java.lang.String	**getName**()
int	**getParking**()
int	**getRoomNum**()
java.lang.String	**getSource**()
java.lang.String	**getTel**()
java.lang.String	**getView**()

Fig. 40. Documentation form of interface for Client class

```java
/**
 * The is an interface, has abstract methods of DataBase
 */
public interface DataBaseSS
{

    //mutators
    public void setName(int dist, int client, String changeTo);
    public void setGender(int dist, int client, char changeTo);
    public void setAge(int dist, int client, int changeTo);
    public void setTel(int dist, int client, String changeTo);
    public void setMail(int dist, int client, String changeTo);
    public void setSource(int dist, int client, String changeTo);
    public void setIfSoldTo(int dist, int client, boolean changeTo);
    public void setIfGarden(int dist, int client, boolean changeTo);
    public void setView(int dist, int client, String changeTo);
    public void setParking(int dist, int client, int changeTo);
    public void setLocker(int dist, int client, int changeTo);
    public void setRoomNum(int dist, int client, int changeTo);
    public void setAddInfo(int dist, int client, String changeTo);
```

```java
    //accessors
    public String getName(int dist, int client);
    public char getGender(int dist, int client);
    public int getAge(int dist, int client);
    public String getTel(int dist, int client);
    public String getMail(int dist, int client);
    public String getSource(int dist, int client);
    public boolean getIfSoldTo(int dist, int client);
    public boolean getIfGarden(int dist, int client);
    public String getView(int dist, int client);
    public int getParking(int dist, int client);
    public int getLocker(int dist, int client);
    public int getRoomNum(int dist, int client);
    public String getAddInfo(int dist, int client);

    //writing and reading the text file
    public void toBeWritten();
    public void textToRead();

    //creating district
    public void createDistrict(String name);
    //creating client
    public void createClient(int district, String name, char gender, int age,
                String tel, String mail, String source, boolean ifSoldTo,
                boolean ifGarden, String view, int parking, int locker,
                int roomNum , String addInfo);
```

Fig. 41. Interface for DataBase class



Interface DataBaseSS

public interface **DataBaseSS**

The is an interface, has abstract methods of DataBase

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
void	**createClient**(int district, java.lang.String name, char gender, int age, java.lang.String tel, java.lang.String mail, java.lang.String source, boolean ifSoldTo, boolean ifGarden, java.lang.String view, int parking, int locker, int roomNum, java.lang.String addInfo)
void	**createDistrict**(java.lang.String name)
java.lang.String	**getAddInfo**(int dist, int client)
int	**getAge**(int dist, int client)
char	**getGender**(int dist, int client)
boolean	**getIfGarden**(int dist, int client)
boolean	**getIfSoldTo**(int dist, int client)
int	**getLocker**(int dist, int client)
java.lang.String	**getMail**(int dist, int client)
java.lang.String	**getName**(int dist, int client)
int	**getParking**(int dist, int client)
int	**getRoomNum**(int dist, int client)
java.lang.String	**getSource**(int dist, int client)
java.lang.String	**getTel**(int dist, int client)

Fig. 42. Part of documentation form of interface for dataBase class

words: 1062

Sources:

Awad, M., n.d. *Java Swing Tutorial: Examples To Create GUI*. [online] Guru99.com. Available at: <https://www.guru99.com/java-swing-gui.html> [Accessed 20 January 2020].

Docs.oracle.com. n.d. *Actionevent (Java Platform SE 7)*. [online] Available at: <https://docs.oracle.com/javase/7/docs/api/java/awt/event/ActionEvent.html> [Accessed 15 February 2020].

Krejcha, J., 2012. *How To Write A Unit Test?*. [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/8751553/how-to-write-a-unit-test> [Accessed 23 February 2020].

Rayapati, P., n.d. *Singleton Class In Java - Geeksforgeeks*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/singleton-class-java/> [Accessed 24 February 2020].

Tutorialspoint.com. n.d. *Java - Interfaces - Tutorialspoint*. [online] Available at: <https://www.tutorialspoint.com/java/java_interfaces.htm> [Accessed 15 December 2019].