**Documentation**

**“Take-a-Room” Application**

Below is presented, in details, a server-client application made to ease the reservation process for a hotel room and give better access and control over those for the hotel staff.

The system consists of 2 separate applications, a server and a client, that communicate between them through the TCP protocol, using sockets. Each application can be started though its own .jar file. Below is the description of both components:

**The Server:**

This is the part of the system that handles the data managing, saving and processing. The data is received from the client. The user interface is done with the help of java package for graphics and design, called javafx. The functionality in the back uses: threads, files handling ( reading / writing ) and the TCP protocol with sockets. This application is used by the hotel staff to manage and control the reservations.

Classes:

* *Server.java* -> has the main thread that waits for clients to connect and creates a separate thread for every client connected. It also has all the main functionality like: starts the server on a predetermined port and ip, initiates the memory by reading the files, makes a reservation, checks for available rooms, deletes a reservation and checks the status of a reservation.
* *ServerWorker.java* -> an object of this class is created for every client that connects to the server. It handles the communication with the client using DataInputStreams and DataOutputStreams
* *ServerDesign.java* -> handles everything related to the user interface. Creates all the scenes with all the elements on them and all the actions resulted from user input. A scene has a layout of BorderPane that has its middle set to a GridPane
* *Packet.java* -> the client and the server have predetermined packet structure that they use to communicate. The packet structured is: PacketType – PacketLenght – Data
* *ServerResChecker.java* -> is a separate thread that checks the time, and every 24 hours checks if a reservation has passed so it deletes it or if a reservation is closer than 24 hours so it announces it
* *Rooms.java , Reservations.java, Person.java* -> all of those are classes that help memorizing reservations information. The rooms and the details about them: number, price per room, number of places in room ; the reservation details: arrive date, leave date, price ; the personal details: name, cnp, phone number, email and payment details.

Main methods:

* ServerWorker -> public void run() -> consists of a continuous while loop that waits for a message from the client. After it receives one, it reassembles the packet from the message and according to its type, makes an action. Based on the results of that action it creates the new packet and sends its back to the client
* Server -> public String searchReservation(String data) -> splits the data into information that it is used to search if a room is available. That information is the arrive date, the leave date and the number of persons. If it founds a room with the correct number of persons and the reservation period free, it returns the room number and the price for the reservation, if it doesn’t find a room it return “none”
* Server -> public int makeReservation(String data) -> based on the data it gets, it makes a reservation in the room found before with the searchReservation() method, adds the new reservation, with the details, to that room’s reservation list and it also adds the personal information to its respective list and generates an unique code that helps identifying the reservations. On success it returns the code, on fail it returns -1
* Server -> public String checkReservation(String data) -> the data is split into name and code and based on those it checks to see if they match to a reservation. If they do, it returns “Yes”+ the reservation details, if they don’t it returns “No”
* Server -> public int deleteReservation(String data) -> based on name and code, it deletes the reservation that matches them. It returns 1 on success and -1 on fail.

User Interface:

* Menu scene -> it has a textArea object where all the actions that the clients and the server take are printed, and 2 buttons: one for exit and one for switching to the other scene.
* Manage scene -> it has a ListView object that stores all the reservation showed by name – code, a textArea object to print all the information for the reservation clicked in the list, a textFiled to input a name, a code or a room number to search the list by and 3 buttons: one for the search, one to delete the selected reservation and one to return back to the menu scene

**The Client:**

This is the part of the system that the people who want to make a reservation to that hotel use. It gets the input from the user and takes actions according to it. The user interface is done using the same technology as the server, meaning javafx. The functionality in the back is simpler than the server’s because it has only one main thread that handles everything. The communication with the server is done with sockets using the same packet system as the server.

Classes:

* *Client.java* -> the main function for this class is the communication with the server. The user input data is sent to the methods in here from the ClientDesign. Here, that data is assembled into packets and sent to the server. Then the client waits for an answer and sends that answer back to the ClientDesign to take a decision accordingly
* *ClientDesign.java* -> it holds all the user interface creation and managing. All the scenes with all the objects on them and all the actions that happen when a user does something are made here. A scene is created with a BorderPane layout that has a GridPane in the middle. This class also handles some of the user input verification and the decisions that are made based on that input
* *Packet.java* -> is the same class as the one for the server. Its purpose is to structure the packet for an easier method of communication with the server
* *MyConnectionException.java, MyCommunicationException.java* -> are custom made exceptions to help with the cases when the server is not working or the communication between the two is broken

Main Methods:

* public int searchRoom(), int makeReservation(), String searchReservation(), int deleteReservation() all work in the same way. They get the user input, they assemble the packet and send it to the server waiting for an answer. When they get it, they send it to the ClientDesign
* public boolean checkDatesAndPersonCount(), checkUserData() both check the user input to see if it matches the rules. The first one checks the arrive date, leave date, and number of persons and the other one checks the cnp, phone number, email etc. If errors are found they print them in AlertBox objects and return false, otherwise they return true

User Interface:

* Menu scene -> has the hotel title and 4 buttons: one to exit, one to see the hotel offers and details, one for reservation checking and one for reservation making. When clicked, all of them change the current scene to their respective one using the private void setCurrentScene(Scene scene) method
* HotelDetails scene -> is a simple scene containing pictures and information, shown in labels, with the hotel offers. It also has a button that takes you back to the Menu scene
* MakeReservation scene -> has 3 textFields for the arrive date, 3 for the leave date and one for the number of people. All of them have description labels near. It also has 3 buttons: one to return to the Menu scene, one to search for a room based on the data in the textFields and one that becomes active only if a room was found. In addition, there is an empty label that becomes visible only after a search was made and it shows the result in either green on success or red on fail
* ConfirmReservation scene -> has 7 textFields for personal information, all with description labels and 2 buttons: one to go back and one to confirm the reservation.
* ManageReservation scene -> has 2 textFields for the name and the code, both with description labels, a button that searches for the reservation based on the data in the 2 textFields, 5 labels to print the information about the reservation if one was found, or the “No” text if no reservation was found. Under all these, there are 2 buttons: one to go back and one that becomes active only when a reservation was found. Pressing this one, deletes that reservation.

**Tests:**

Client Tests:

1. Testing user input verification for reservation dates and number of persons:

1. Input: cg.checkDatesAndPersCount(12, 14, 10, 10, 2019, 2019, 3);

Expected Output: true;

Output: true;

Cause: Everything is correct;

1. Input: cg.checkDatesAndPersCount(0, 14, 10, 10, 2019, 2019, 3);

Expected Output: false;

Output: false;

Cause: arrive day value is lower than 1;

1. Input: cg.checkDatesAndPersCount(12, 14, 10, 10, 2019, 2019, 6);

Expected Output: false;

Output: false;

Cause: Number of persons is bigger than 4;

1. Input: cg.checkDatesAndPersCount(12, 14, 10, 10, 2019, 2018, 3);

Expected Output: false;

Output: false;

Cause: leave year value is before the current year;

1. Input: cg.checkDatesAndPersCount(12, 33, 10, 10, 2019, 2019, 3);

Expected Output: false;

Output: false;

Cause: leave day value is bigger than 31;

1. Input: cg.checkDatesAndPersCount(12, 14, 10, 10, 2020, 2019, 3);

Expected Output: false;

Output: false;

Cause: arrive year value is bigger than leave year value;

2. Testing user input verification for reservation details:

1. Input: cg.checkUserData("CodilaOvidiu", "1234567890123", "ovi.codila@gmail.com", "0771339028", "1234567890123456", "657");

Expected Output: true;

Output: true;

Cause: Everything is correct;

1. Input: cg.checkUserData("", "1234567890123", "ovi.codila@gmail.com", "0771339028", "1234567890123456", "657");

Expected Output: false;

Output: false;

Cause: Name is empty or null;

1. Input: cg.checkUserData("CodilaOvidiu", "12347890123", "ovi.codila@gmail.com", "0771339028", "1234567890123456", "657");

Expected Output: false;

Output: false;

Cause: CNP is too short;

1. Input: cg.checkUserData("CodilaOvidiu", "1234567890123", "ovi.codila@gmail.com", "0771339x028", "1234567890123456", "657");

Expected Output: false;

Output: false;

Cause: Phone number contains letters;

1. Input: cg.checkUserData("CodilaOvidiu", "1234567890123", "ovi.codila@gmail.com", "0771339028", "1234567890123456", "6570");

Expected Output: false;

Output: false;

Cause: CVV is too long;

1. Input: cg.checkUserData("CodilaOvidiu3", "1234567890123", "ovi.codila@gmail.com", "0771339028", "1234567890123456", "657");

Expected Output: false;

Output: false;

Cause: Name contains numbers;

3. Testing client connecting to the server

1. Input: cl.startClient(); and server is OFF;

Expected Output: MyConnectionException;

Output: MyConnectionException;

Cause: Server is OFF and the client tried to connect to it;

1. Input: cl.startClient(); and server is ON;

Expected Output: false;

Output: false;

Cause: Server is ON so the client connected to it thus failing the assertion;

4. Testing the method that requests a reservation to be deleted

1. Input: cl.deleteReservation("Codila Ovidiu", "401182"); server ON;

Expected Output: -1;

Output: -1;

Cause: The reservation doesn’t exist;

1. Input: cl.deleteReservation("Codila Ovidiu", "401182"); server OFF;

Expected Output: MyConnectionException;

Output: MyConnectionException;

Cause: server is OFF;

Server Tests:

1. Testing memory initialization

1. Input: sv.initMemory();

Expected Output: true;

Output: true;

Cause: Everything went well and the memory was initialized;

2. Testing reservation checking

1. Input: sv.checkReservation("Codila Ovidiu/401016");

Expected Output: "Yes" + details;

Output: "Yes" + details;

Cause: Reservation was found;

1. Input: sv.checkReservation("Ion Popescu/301089");

Expected Output: "No";

Output: "No";

Cause: Reservation was not found;

3. Testing code checking when a new one is created

1. Input: sv.codeExists(401016);

Expected Output: true;

Output: true;

Cause: The code was found in the database;

1. Input: sv.codeExists(301088);

Expected Output: false;

Output: false;

Cause: The code was not found in the database;

1. Input: sv.codeExists(401189);

Expected Output: false;

Output: false;

Cause: The code was not found in the database;