# Cloud Surf Inn: Technical Documentation

**Submitted To:** Software Engineering

**Submitted On:** 04/02/2021

**Submitted By:** Group 8

**Website:** https://sites.google.com/view/cloud-surf-inn/home?authuser=0

**Github:** https://github.com/Software-Engineering-Team-8/Cloud-Surf-Inn

**Group Members:** Chynna Walsh, Sidonia Mohan, Ryder Morrell, JP Dangler, Juan Escudero, Zach LeMunyon, Chris Kline, Bharath Selvaraj, Jakub Vogel, Sebastian Matiz

**<u>Table of Contents</u>**

## 1. Housekeeping

In Cloud-Surf-Inn/1_code/CSIv1.44/src/cloud_surf_inn, you will find the CleaningNode.java, HKPRHeap.java, and HPHeap.java files that provide the foundation for the implementation of Cloud Surf Inn's housekeeping system. All of these classes have constructors, getters and setters, and methods that allow for the creation and manipulation of CleaningNodes within the heaps. Naturally, HKPRHeap and HPHeap classes are priority queues that consist of CleaningNodes, specifically minheaps, that organize the housekeeping requests that customers solicit. Minheaps were chosen as the data structure used to organize housekeeping requests because the most urgent request (the request happening the soonest and thus with the lowest priority) would be at the top of the heap whereas less urgent requests (requests happening "later" and thus with higher priorities) would always be further down. Minheaps allow requests to be organized based on urgency/recency relative to the current time which allows housekeepers to conveniently service more urgent requests that will always be at the top of the minheaps. Both heap classes implement Java's built-in Priority Queue class, and thus inherit some of the methods that the Priority Queue class offers. Consequently, the HKPRHeap and HPHeap classes need a "natural ordering" with which to arrange themselves with a minheap ordering. This "natural ordering" is dictated by a priority calculated by the CleaningNode class that utilizes the Java Time class to calculate the urgency of a request. The priority is calculated whenever a node is created (with the static priorityCalculator method), and the priority itself is defined as the number of hours between the current hour and the hour on the day which the request is to be serviced. For instance, if it is 6 P.M. Friday, and a customer schedules a housekeeping request for 6 P.M. Saturday, the priority of this CleaningNode would be 24 because there are 24 hours between the current time and the scheduled time. It is important to note that minutes are not considered when calculating priority and the CleaningNode is comparable to itself in order for CleaningNodes to be compared to each other with respect to their priorities. There are also certain constraints on creating CleaningNodes; CleaningNode cannot be scheduled within two hours of the current time and they can only be scheduled up to a week in advance.

As stated before, the HPHeap and HKPRHeap are priority queues. The distinction between the two heaps is that the HPHeap contains all housekeeping requests made by customers, and there will only be one active HPHeap at any given time. On the other hand, the HKPRHeap contains the requests accepted by a housekeeper and there will be as many instances of the HKPRHeap class as there housekeepers on duty. Both of the heaps have overloaded methods to addNodes, removeNodes, and deleteNodes (depending on which operations need to be performed on the heaps) in addition to methods that are inherited directly from the Priority Queue class. The distinction between the removeNodes and deleteNodes methods is that the removeNodes returns the node that was removed whereas deleteNodes delete the node(s) that were selected.

## 2. Mobile Key

In “Cloud-Surf-Inn-main\Demo1Folder\CSIv1.44\bin\cloud_surf_inn” you will find file “oneTimePassword.class”. This file contains that code that will run that generates a random 10 alphanumeric pin that the user can use to access a room. This will run when the user logs in and selects the “book a room” button. This will call the functions roomMatcher and ontTimePassword. Once the roomMatcher function is called, and all conditions are met, then a key will be outputted into the GUI that will allow access to the room for the specific user. This code is used again when generating the masterkey. Once the user selects the manager tab and signs in with proper credentials, in the managers GUI there will be a randomly 10 digit alphanumeric pin that will allow access to all rooms displayed on the tab.

## 3. Power Control

Summary:

In **Cloud-Surf-Inn/1_code/CSIv1.44/src/cloud_surf_inn/ TCPsock.java** you will find the code that handles operations which include check-in, check-out, change temp and show. This file is considered the server side in the thermostat and GUI connection. When the user selects Check-In the activateControl function is called and a connection with the thermostat and GUI is established. A few things to note: The port number match that provided from the client and server side, and the IP address used to establish the connection should be the IPv4 address of the machine running Cloud Surf Inn. As soon as the connection is established the thermostat goes into comfort mode. The user can now read data and write data to the thermostat given what the GUI provides. Once the user checks out the connection is closed, and temperature turns off while we wait to establish another connection.

Hardware used: Computer to run Cloud Surf Inn.

In **Cloud-Surf-Inn/tempsensor (2).py** you will find the code that handles operations which include check-in, check-out, change temp and show. This file is considered the client side in the thermostat and GUI connection. When the user selects Check-In, and the given port number and IPv4 address is provided, a connection is established and the Raspberry Pi reads messages from the GUI to do different functions. For example if the customer selects check-in a "booked" message from the GUI to the Pi is sent indicating that comfort mode is activated. Different messages provide the Pi with different instructions for temperature.
Hardware used: Raspberry Pi 3, DHT Temperature and Humidity Sensor

*Ethernet Cable is used to provide a connection between the Raspberry Pi and the Computer running Cloud Surf Inn

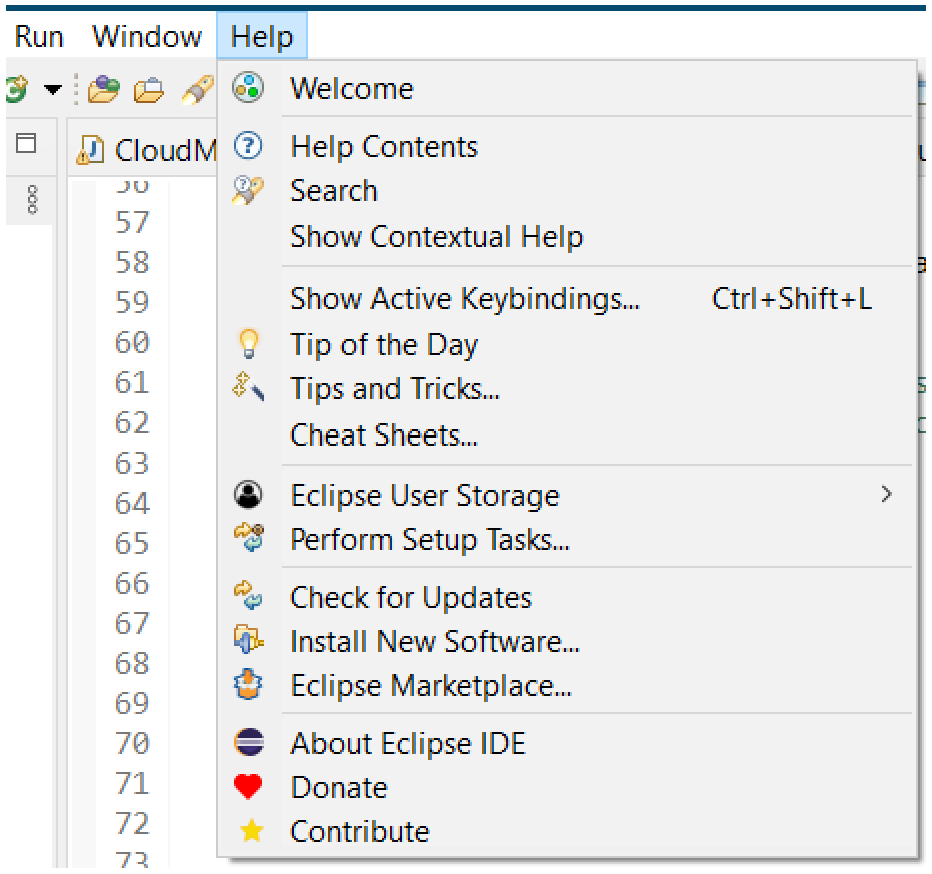
## 4. Room Matcher

In directory “Cloud-Surf-Inn-main\Demo1Folder\CSIv1.44\bin\cloud_surf_inn” you will find file roomMatcher.class. This file contains the code behind the function “roomMatcher” that is called when the user clicks the “book a room” button. This program is a simple algorithm that takes in the room database and user preferences as inputs. From these inputs, it creates a copy database of integers “datafake” to be run through and matched by a ranking system with the user preferences. If the user preference value matches database value for the given amenity, the room has its preference rank increased by 2. If the room has the correct amount of beds, it is also given an increase of 2. However, if the room has more than the required beds, it also gets 1 increment, to ensure that we prioritize the proper amenities. Once all rooms are scanned, we then scan them again to find the highest preference rank, and output that room to our customer and set it to be booked.

## 5. Running the Application

Provided below are the steps required to run our program. Before following the steps, please make sure that the following prerequisites are installed on your system: Windows OS (7/8/10), Eclipse IDE version 2020-03 or newer (older versions have the potential to run, however this tutorial is intended for the given version), Java 8, and a stable internet connection.

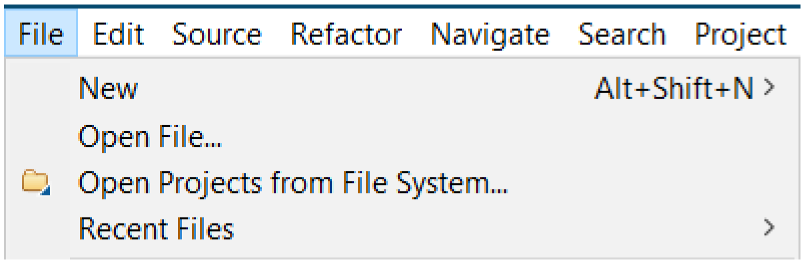
**1.** Open up Eclipse IDE to the main screen. The project explorer, console, workspace, and toolbar should all be visible. Go to the toolbar and click the "help" button located all the way to the right. A drop down menu will appear: click on the "Eclipse Marketplace…" option.



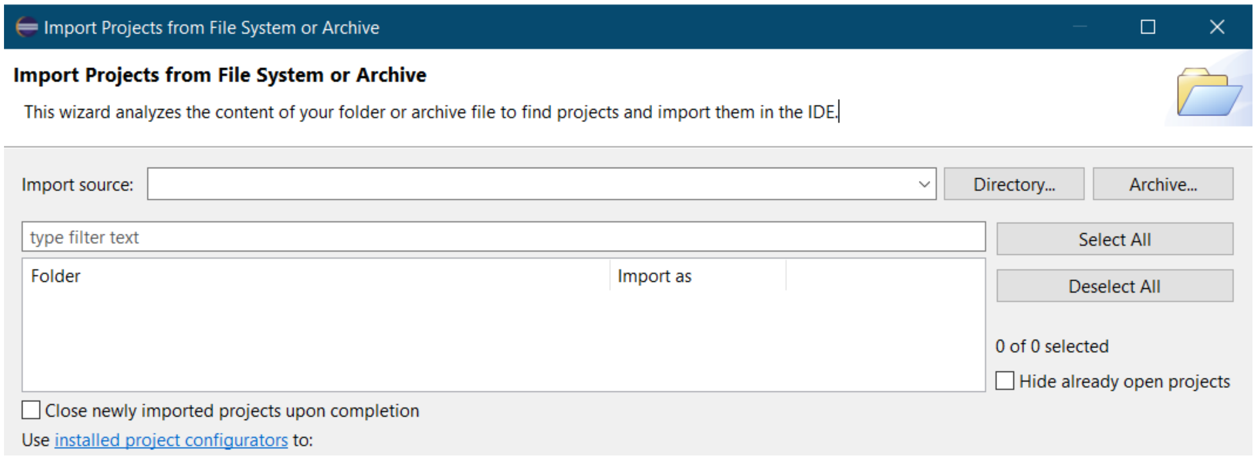
From the eclipse marketplace you need to search for and then install "WindowBuilder 1.9.5". Eclipse IDE should handle the installation if a stable internet connection is present.

**2.** Download a .zip file from our team GitHub which can be found at: https://github.com/Software-Engineering-Team-8/Cloud-Surf-Inn. Extract the .zip file into a secure location that can be accessed by Eclipse IDE. Go to your Eclipse IDE toolbar and select

"File" located all the way to the left. From here you can select the option "Open Projects from File System…".


File Edit Source Refactor Navigate Search Project
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files

Once this option is selected, a window will appear that will allow you to open a project from a selected registry.


Import Projects from File System or Archive

**Import Projects from File System or Archive**

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: [ ] Directory... Archive...

type filter text — Select All

Folder | Import as — Deselect All

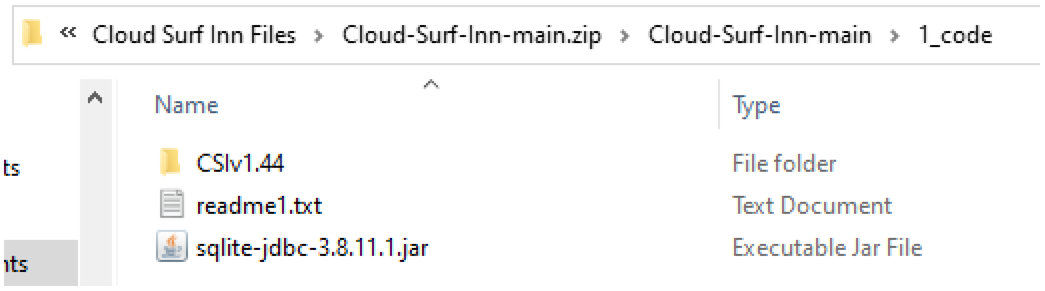
0 of 0 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

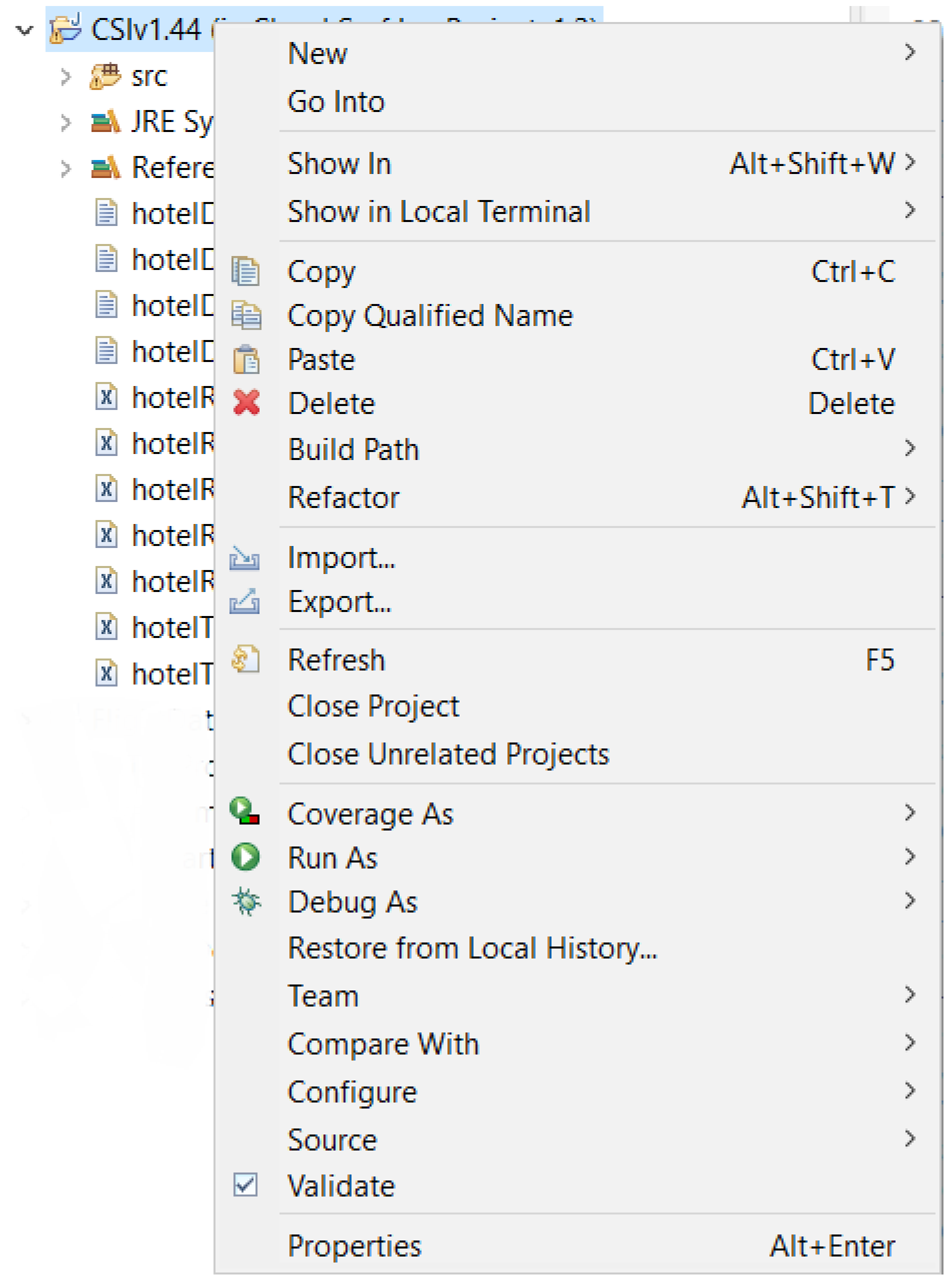
Click on the "Directory…" button and then find the extracted contents of the .zip file that was downloaded earlier. From this file, navigate to the directory "Cloud-Surf-Inn-main/1_code" where you should see a "CSIv1.44" folder, readme.txt, and sqlite-jdbc-3.8.11.1.jar.


« Cloud Surf Inn Files › Cloud-Surf-Inn-main.zip › Cloud-Surf-Inn-main › 1_code

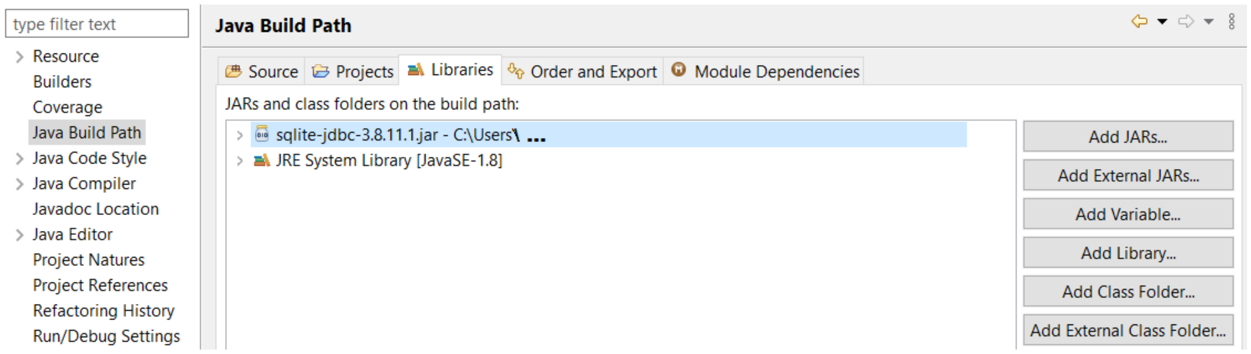
| Name | Type |
|---|---|
| CSIv1.44 | File folder |
| readme1.txt | Text Document |
| sqlite-jdbc-3.8.11.1.jar | Executable Jar File |

Select the folder "CSIv1.44" and then click the "Finish" button at the bottom of the screen. The project should now be visible under the "Project Explorer" located to the left of the main screen.

**3.** Now that the project is available to be used by the Eclipse IDE, we have to add an external .jar file in order for the project to run correctly. From the "Project Explorer" right click on the "CSIv1.44" project that was added in the previous step. A dropdown menu will appear which will have the option "Properties" at the bottom.

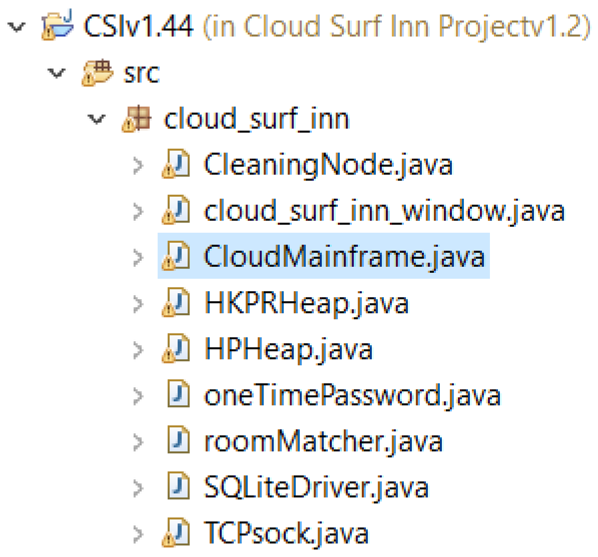


A window will appear that has various options to select from on the left side. Select the option "Java Build Path" and then select "Libraries" located in the middle of the window.

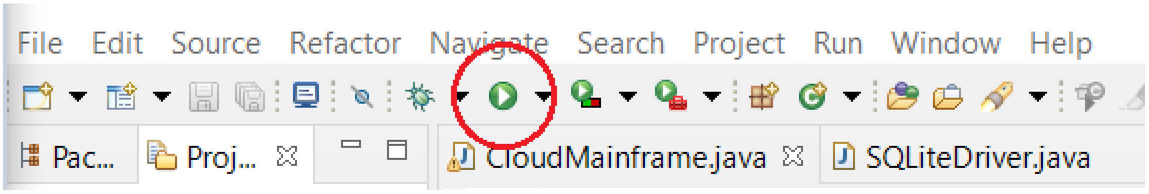


Select the "Add External JARs…" button on the right side of the window and then once again find the extracted contents of the .zip file that was downloaded. Go to the directory "Cloud-Surf-Inn-main/1_code" (same as before) and then select the "sqlite-jdbc-3.8.11.1.jar" file. Once this is done, click the "Apply and Close" button at the bottom of the screen.

**4.** We are nearly finished setting up. Go to the "Project Explorer" and navigate to the directory "CSIv1.44/src/cloud_surf_inn" where a handful of .java files should be located. Proceed to double click on the file "CloudMainframe.java" which should then open the file in the workspace on the main screen.



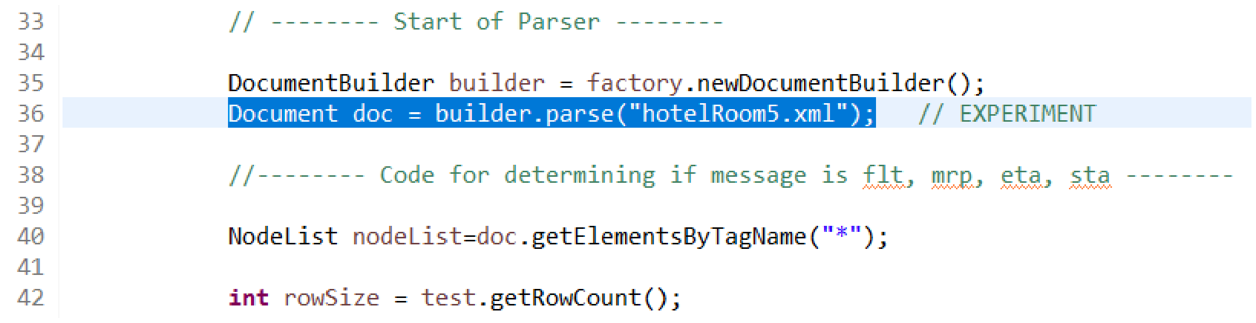
Now you can run our project by clicking on the green run button located below the toolbar's "Navigate" button.


File Edit Source Refactor Navigate Search Project Run Window Help
Pac...
Proj...
CloudMainframe.java
SQLiteDriver.java

The program will now run and display our GUI with all its features. Enjoy!

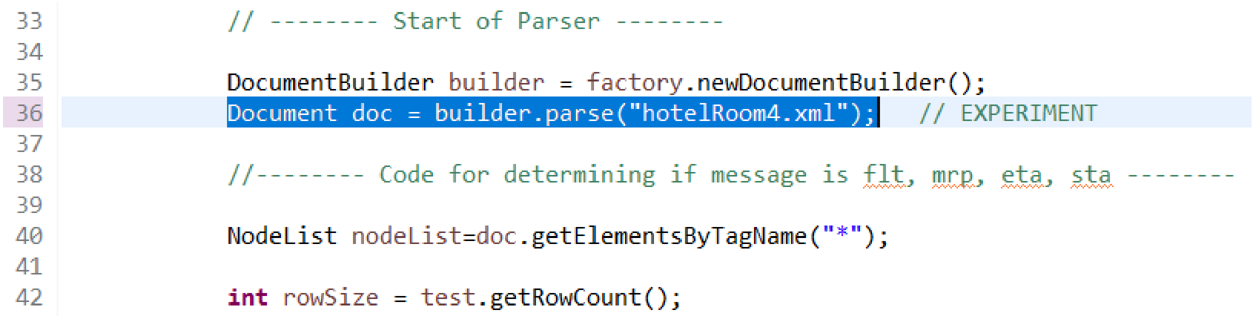
## Adding Additional Database Entries:

In order to add additional entries (hotel rooms) to the database we must do so by modifying a single line of code in the “CloudMainframe.java” file. The statement “Document doc = builder.parse(“hotelRoom5.xml)” decides what .xml file will be written to the database so it can be used later.



```
            // -------- Start of Parser --------

            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse("hotelRoom5.xml");    // EXPERIMENT

            //-------- Code for determining if message is flt, mrp, eta, sta --------

            NodeList nodeList=doc.getElementsByTagName("*");

            int rowSize = test.getRowCount();
```

In order to change the .xml file that is being inputted, we must change the number that comes after “hotelRoom”. For example, “hotelRoom5” would instead have to be “hotelRoom1”, “hotelRoom2”, etc.



```
            // -------- Start of Parser --------

            DocumentBuilder builder = factory.newDocumentBuilder();
            Document doc = builder.parse("hotelRoom4.xml");    // EXPERIMENT

            //-------- Code for determining if message is flt, mrp, eta, sta --------

            NodeList nodeList=doc.getElementsByTagName("*");

            int rowSize = test.getRowCount();
```

There are currently 5 different hotel room entries that are provided with our project, although more are planned to be added in the future. Once multiple rooms are added to the database, they will each provide data that is visible via the GUI.