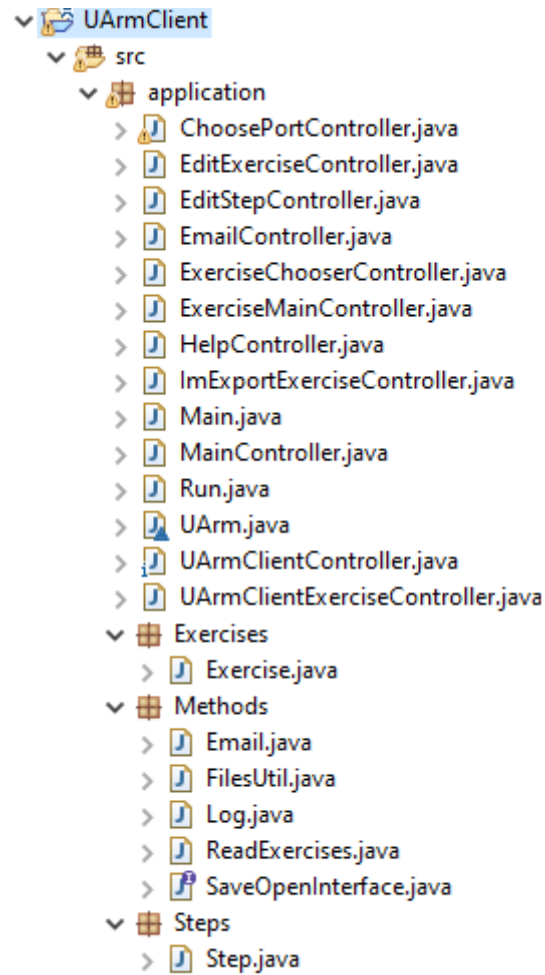


Criterion C: Product Development

Figure 1- All Classes



Some Techniques used:

- Development of Methods
- Manipulation of sequential files
- Use of Graphic User Interface
- Establishing connection between user and uArm
- Static methods and variables
- Modular Programming

Methods

The following method is essential to establish a connection between the user and the **uArm**.

```
class UArm {
    SerialPort serialPort;
    int LimitXMin = -300;
    int LimitXMax = 300;
    int LimitYMin = 80;
    int LimitYMax = 300;
    int LimitZMin = 25;
    int LimitZMax = 250;

    public UArm(String port) throws SerialPortException {
        serialPort = new SerialPort(port);
    }

    // Makes a connection with the Uarm, with all the information that is used to send information
    public void setPort() {
        try {
            System.out.println("Port opened: " + serialPort.openPort());
            System.out.println("Params setted: " + serialPort.setParams(SerialPort.BAUDRATE_115200,
                SerialPort.DATABITS_8, SerialPort.STOPBITS_1, SerialPort.PARITY_NONE));
        } catch (SerialPortException e) {
        }
    }
}
```

This opens the port and sets the parameter of the **uArm** for the speed of transmission and for reading data using a simple third party library called *JSSC* (Java Simple Serial Controller) because it can easily establish a connection without using complex algorithms.¹

```
import jssc.SerialPort;
import jssc.SerialPortException;
```

The following method makes the arm move and checks the limits set.

```
// sends information to the Uarm, to make it move
void goToPosition(int x, int y, int z) throws Exception {
    // Checks if it is going over the limit, if so, change coordinates
    // that can be read by the arm
    if (x > LimitXMax) {
        x = LimitXMax;
        hitLimit();
    }
    if (x < LimitXMin) {
        x = LimitXMin;
        hitLimit();
    }
    if (y < LimitYMin) {
        y = LimitYMin;
        z += 100;
        hitLimit();
    }
    if (y > LimitYMax) {
        y = LimitYMax;
        hitLimit();
    }
}
```

¹ "jSSC (Java Simple Serial Connector)." *javalibs*, javalibs.com/artifact/org.scream3r/jssc. Accessed 8 Jan. 2021.

```

    if (z < LimitZMin) {
        if (y <= 160) {
            z = LimitZMin + 10;
        } else {
            z = LimitZMin - 20;
        }
        hitLimit();
    }
    if (z > LimitZMax) {
        z = LimitZMax;
        hitLimit();
    }
    // the F would be the speed in mm/min
    String run = "#1 G0 X" + x + " Y" + y + " Z" + z + " F60\n";
    try {
        serialPort.writeBytes(run.getBytes());
        TimeUnit.MILLISECONDS.sleep(700);

    } catch (SerialPortException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

The string is converted into bytes and read by the arm, to go to a certain coordinate. Additionally, the Enumerated type **TimeUnit** creates an interval before the next step for the user to watch the steps taken.² These limits were developed through trial and error, so no unnecessary movements are executed (shown in the video) as it calls the next method to make the arm beep, when a limit's hit to inform the user, for user-friendliness reasons.

```

import java.util.concurrent.TimeUnit;

// Makes a beep when the limit is hit
void hitLimit() throws Exception {
    String limit = "#1 M210 F2800 T0.5\n";
    try {
        serialPort.writeBytes(limit.getBytes());

    } catch (SerialPortException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

The following method located in the **UArmClientController** executes the list of Steps.

```

// Makes the Uarm Move, by calling the Run class
@FXML
void btRun_clicked(ActionEvent event) throws Exception {
    save();
    Run run = new Run(uarm);
    run.getClass();
}

```

² "Enum TimeUnit." Oracle, docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html. Accessed 8 Jan. 2021.

The **Run** class inherits this controller, for organizational reasons, as it makes a *linear search* for each direction and corresponding distance in the order the user selected.

```
public class Run extends UArmClientController {

    UArm uarm;

    // Sends information to the Arm so that it can move to the inputed direction
    public Run(UArm u) throws IOException, Exception {

        this.uarm = u;
        int x = 1;
        int y = 140;
        int z = 145;
        int temp;

        // reading from the StepsList
        FilesUtil.fileExist("StepsList.txt");
        String content = FilesUtil.readTextFile("StepsList.txt");
        FilesUtil.fileExist("Log.txt");
        Log log = new Log();
        log.printlog(content);
        String[] arr = content.split(";");

        // Searches for each one of the directions and then reads their values
        // that is placed after them
        for (int i = 0; i < arr.length; i++) {
            // Searches for the word Right
            if (arr[i].contains("Right")) {
                String[] arr2 = arr[i].split(":");
                temp = Integer.parseInt(arr2[1]);
                if (temp != 0) {
                    x += temp * 2.5;
                }
                uarm.goToPosition(x, Math.abs(y), z);
            }

            // Searches for the word Left
            if (arr[i].contains("Left")) {
                String[] arr2 = arr[i].split(":");
                temp = Integer.parseInt(arr2[1]);
                if (temp != 0) {
                    x -= temp * 2.5;
                }
                uarm.goToPosition(x, Math.abs(y), z);
            }

            // Searches for the word Front
            if (arr[i].contains("Front")) {
                String[] arr2 = arr[i].split(":");
                temp = Integer.parseInt(arr2[1]);
                if (temp != 0) {
                    y += temp * 1.3;
                }
                uarm.goToPosition(x, y, z);
            }
        }
    }
}
```

```

// Searches for the word Back
if (arr[i].contains("Back")) {
    String[] arr2 = arr[i].split(":");
    temp = Integer.parseInt(arr2[1]);
    if (temp != 0) {
        if ((x > -5 && x <= 5) && (y > 0 && y <= 140)) {
            y -= temp;
        } else {
            y -= temp * 2;
            z += temp / 2;
        }
    }
    uarm.goToPosition(x, y, z);
}
// Searches for the word Up
if (arr[i].contains("Up")) {
    String[] arr2 = arr[i].split(":");
    temp = Integer.parseInt(arr2[1]);
    if (temp != 0) {
        z += temp * 3;
    }
    uarm.goToPosition(x, Math.abs(y), z);
}
// Searches for the word Down
if (arr[i].contains("Down")) {
    String[] arr2 = arr[i].split(":");
    temp = Integer.parseInt(arr2[1]);
    if (temp != 0) {
        z -= temp * 3;
    }
    uarm.goToPosition(x, Math.abs(y), z);
}
}
}
}
}
}

```

Each *if statement* was developed through trial and error to ensure the arm goes to the desired location without error.

The **Log** method shows evidence of completion and a form of backup as it utilizes the **Date**³ and **SimpleDateFormat**⁴ class to inform the user the time of the steps executed.

```

public class Log {
// Prints a log of all the information, directions, when using the Arm
public void printlog(String content) throws IOException {
    Date d = new Date();
    SimpleDateFormat ft = new SimpleDateFormat("E dd/MM/yyyy 'at' hh:mm:ss a zzz");
    String log = ft.format(d) + ": " + content;
    FilesUtil.writeOverTextFile("Log.txt", log);
}
}

```

Figure - Log.txt

Mon 15/02/2021 at 12:44:55 PM GMT: Front:30;Left:40;

³ "Class Date." *Oracle*, docs.oracle.com/javase/7/docs/api/java/util/Date.html. Accessed 8 Jan. 2021.

⁴ "Class SimpleDateFormat." *Oracle*, docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html. Accessed 6 Jan. 2020.

The **pump** can be activated or deactivated.

```
// Setting the state of the pump, activate or not
void pump(int p) throws Exception {
    String pump = "#1 M231 V" + p + "\n";
    try {
        serialPort.writeBytes(pump.getBytes());
    } catch (SerialPortException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

For usability reasons, one click from the **Grab** button changes the pump's status.

```
// Changes the state of the pump
@FXML
void btGrab_clicked(ActionEvent event) throws Exception {
    int pump = 0;

    if (btGrab.getText().equals("Grab")) {
        pump = 1;
        btGrab.setText("UnGrab");
    } else if (btGrab.getText().equals("UnGrab")) {
        pump = 0;
        btGrab.setText("Grab");
    }
    try {
        uarm.pump(pump);
    } catch (SerialPortException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

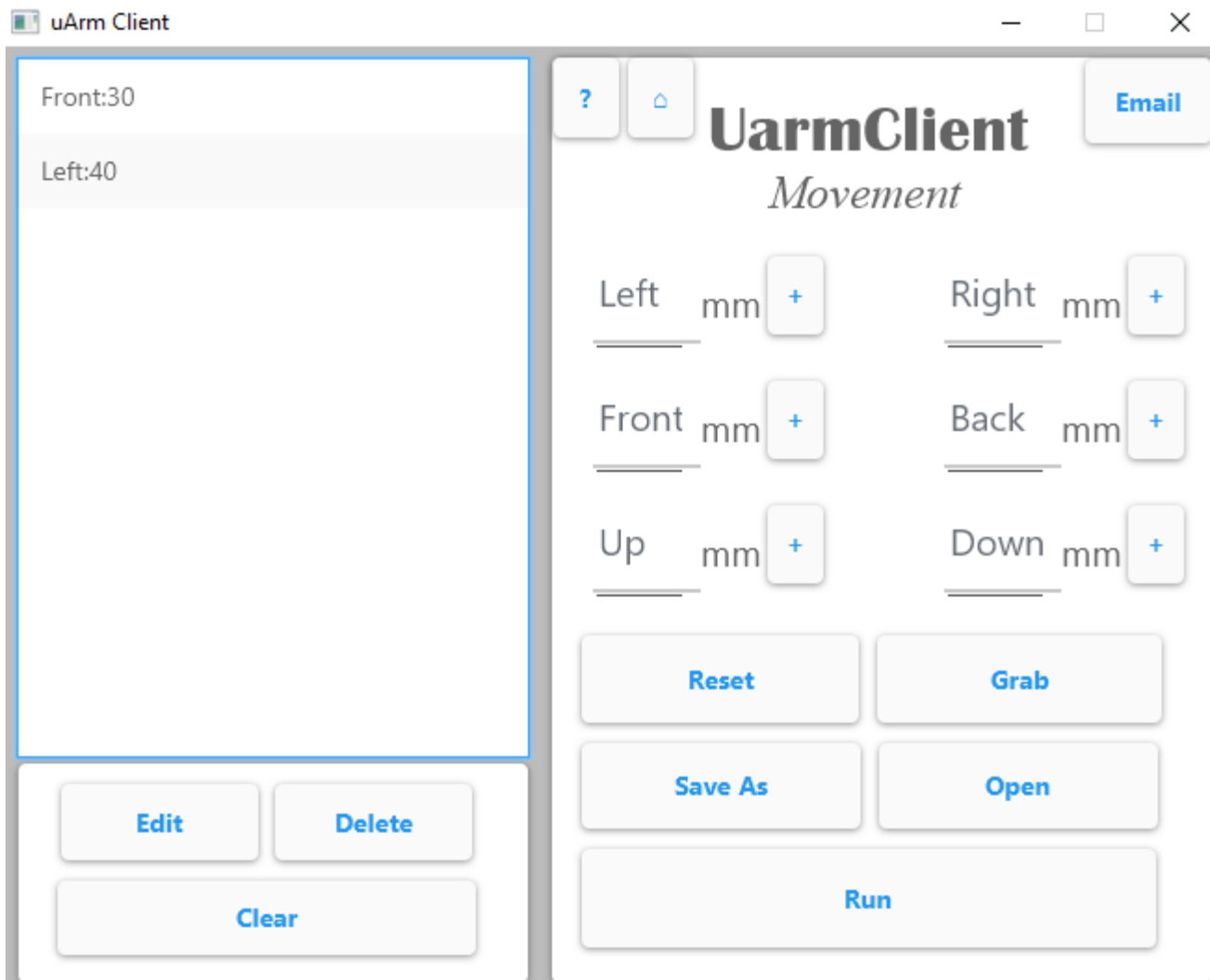
Exception handles are used to maintain the flow of the application, so it doesn't crash or gets interrupted, making the application work more easily.

Graphic User Interface (GUI)

The *GUIs* and *JavaFX* makes the application aesthetically pleasant and facilitates the use of the uArm as it improves the efficiency, functionality, and usability.⁵

The following *GUI* allows simple communication between the user and uArm.

Figure 3- UArmClientController



I used the "+" sign when adding the direction and distance to the list (one of six).

```
// Adds value of going Left to the list
@FXML
void btAddL_clicked(ActionEvent event) {
    Step s = new Step("Left", Integer.parseInt(this.tfLeft.getText()));
    this.stepList.add(s);
    this.tfLeft.setText("");
}
```

⁵ "jfoenix documentation." *JFoenix*, www.jfoenix.com/documentation.html. Accessed 10 Jan. 2021.

For each step created, the direction and distance is set.

```
//this class sets and gets the direction and the distance
public class Step {

    private String dir;
    private int dis;

    public Step(String dir, int dis) {
        this.setDir(dir);
        this.setDis(dis);
    }

    public String getDir() {
        return dir;
    }

    public void setDir(String dir) {
        this.dir = dir;
    }

    public int getDis() {
        return dis;
    }

    public void setDis(int dis) {
        this.dis = dis;
    }

    public String toString() {
        return this.getDir() + ":" + this.getDis();
    }
}
```

Using *encapsulation* and *private variables* protects variables from being easily changed. However, clicking the **Edit** button allows editing.

Figure 4- Editing Distance

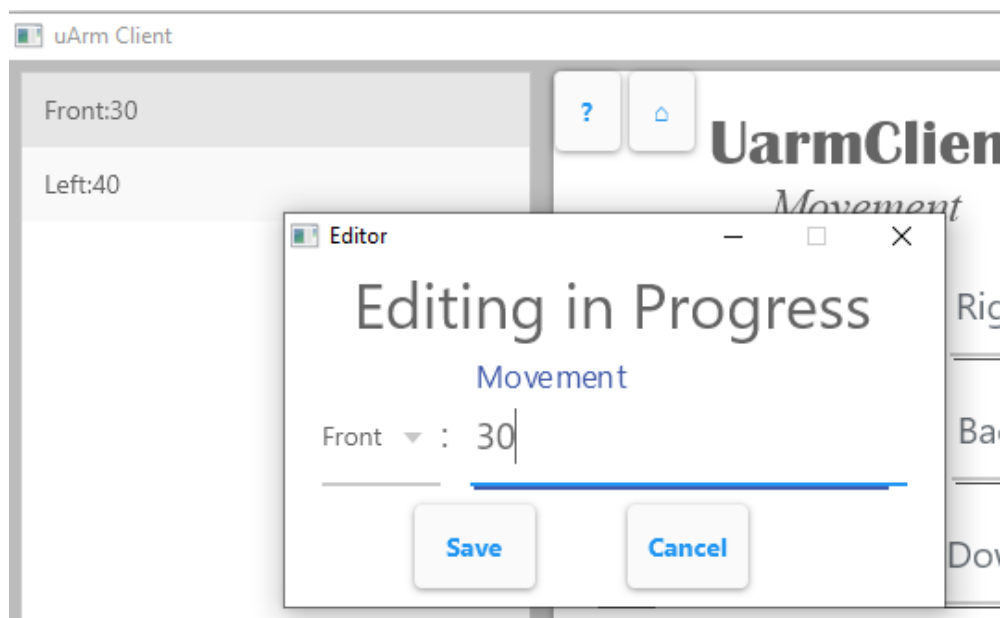
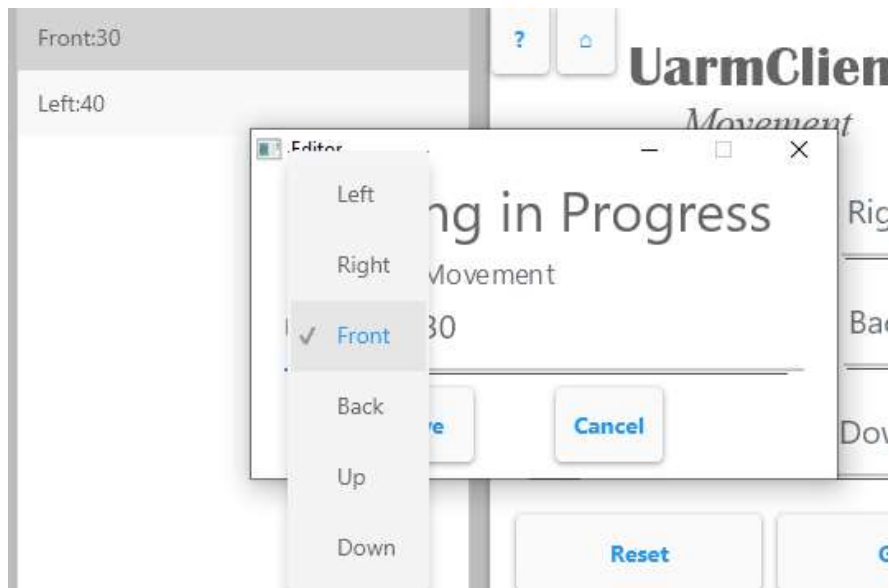


Figure 5-Editing Direction



The **Save** button overwrites the selected step from the list to increase functionality, so the user wouldn't need to delete and add another step.

```
// Saves the information to the list in the UarmClient
@FXML
void btSave_clicked(ActionEvent event) throws IOException {
    EditController.currentStep.setDir(this.cbMove.getSelectionModel().getSelectedItem());
    EditController.currentStep.setDis(Integer.parseInt(this.tfDegree.getText()));
    EditController.parent.refreshList(currentStep);
    Stage stage = (Stage) this.btSave.getScene().getWindow();
    stage.close();
    EditController.parent.save();
}
```

The parent class is the **UArmClientController**.

```
try {
    EditController.currentStep = s;
    EditController.parent = this;

    public class EditController {

        public static Step currentStep;
        public static UArmClientController parent;
```

Using *inheritance* enables me to use the **save** method to save each step into the *current directory* as a .txt called "StepsList" right after editing.

```
// Saves the information as a string in the StepsList file
@FXML
void save() throws IOException {
    String output = "";

    for (Step s : stepList) {
        output += s + ";";
    }

    // this.btSaveAs
    Methods.FilesUtil.writeToFile("StepsList.txt", output);
}
```

The **Save As** and **Open** button creates a File Explorer by using the **FileChooser** class to facilitate user navigation when working with sequential files.⁶

```
// Saves the StepsList text document in the location the user chooses
@FXML
void btSaveAs_clicked(ActionEvent event) throws IOException {
    String output = "";

    for (Step s : steplist) {
        output += s + ";";
    }

    FileChooser fileName = new FileChooser();
    fileName.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Text Document", "*.txt"));
    fileName.setTitle("Save As");
    File file = fileName.showSaveDialog(null);

    SaveAs(output, file);
    save();
}
```

Figure 6- Save As button

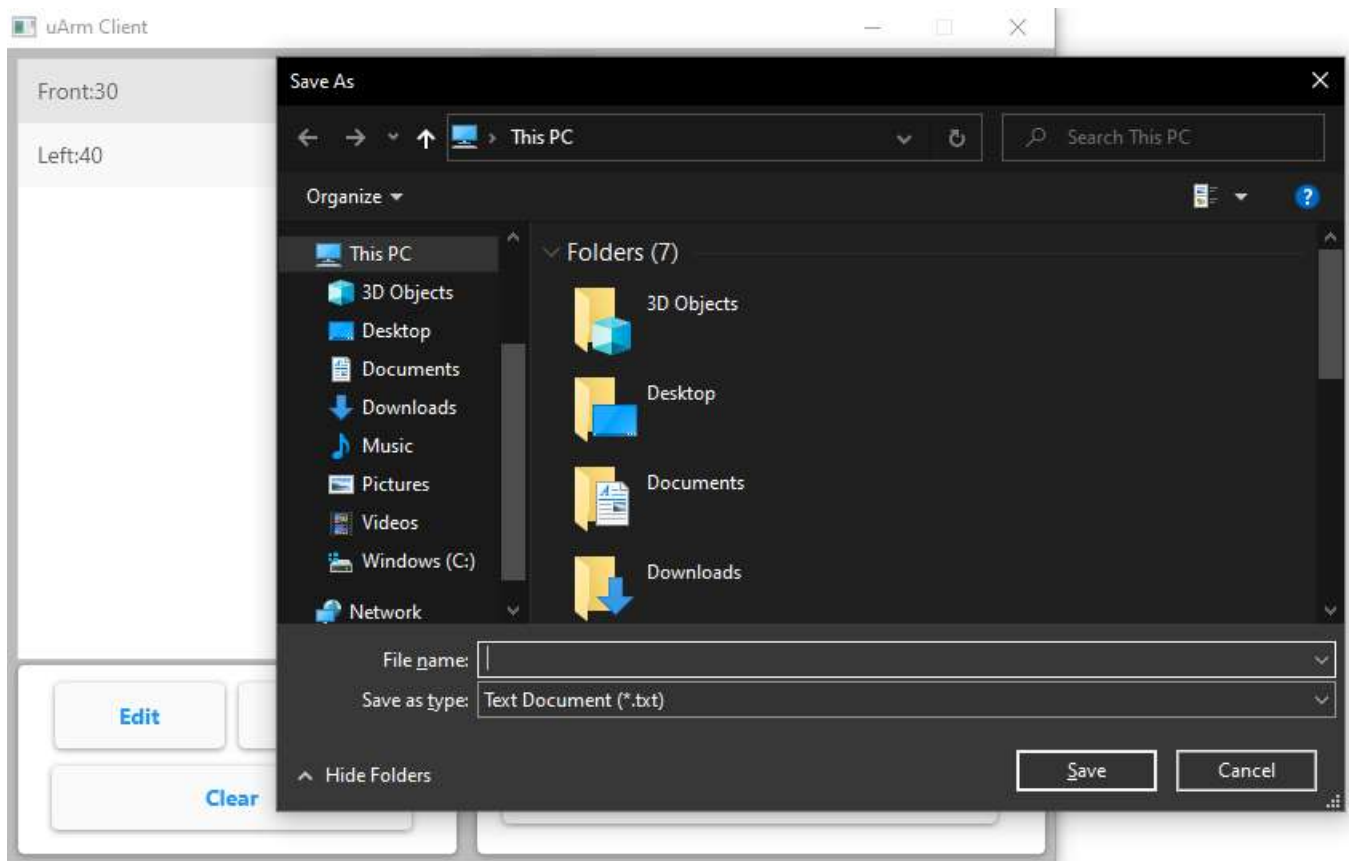



Figure 7- Text Document of Steps List

 StepsList.txt - Notepad

File Edit Format View Help

Front:30;Left:40;

⁶ "Class FileChooser." Oracle, docs.oracle.com/javase/8/javafx/api/javafx/stage/FileChooser.html. Accessed 6 Jan. 2021.

Formatting in CSV (";") is necessary to effectively read files as the *split method* can help read individual steps (direction and distance). Shown when **Open** is clicked.

```
// Opens a text document file that the user chooses
@FXML
void btOpen_clicked(ActionEvent event) throws IOException {
    {
        FileChooser fileName = new FileChooser();
        fileName.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Text Document", "*.txt"));
        fileName.setTitle("Open");
        File file = fileName.showOpenDialog(null);
        String content = Open(file);
        String[] arr = content.split(";");
        stepList.clear();

        for (int i = 0; i < arr.length; i++) {
            String[] arr2 = arr[i].split(":");
            stepList.add(new Step(arr2[0], Integer.parseInt(arr2[1])));
        }
    }
}
```

The **SaveAs** method uses the **PrintStream** class because it is suitable to print various types of data values without throwing *Exceptions*, which helps me better understand future errors.⁷

```
// Inputs the information from the string into the file, using the Interface, SaveOpenInterface
@Override
public void SaveAs(String output, File file) {
    if (file != null) {
        try {
            @SuppressWarnings("resource")
            PrintStream printS = new PrintStream(file);
            printS.println(output);
            printS.flush();
        } catch (FileNotFoundException e) {
            System.out.println("File Not Found");
        }
    }
}
```

The **Open** method uses the **BufferedReader** class to efficiently read strings as it reads a stream of characters from the file after it converts them into a string. An **InputStream** object first needs to be declared as its constructor accepts the **InputStream** object as a parameter to function.⁸

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
```

⁷ "Class PrintStream." *Oracle*, docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html. Accessed 15 Jan. 2021.

⁸ "Class BufferedReader." *Oracle*, docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html. Accessed 10 Jan. 2021.

```
// Outputs the information from the string into the file, using the Interface, SaveOpenInterface
@Override
public String Open(File file) throws IOException {
    if (file != null) {
        InputStream is = new FileInputStream(file);
        @SuppressWarnings("resource")
        BufferedReader buf = new BufferedReader(new InputStreamReader(is));

        String line = buf.readLine();
        StringBuilder sb = new StringBuilder();

        while (line != null) {
            sb.append(line).append("\n");
            line = buf.readLine();
        }
        // Close the input file
        buf.close();

        String fileAsString = sb.toString();
        return fileAsString;
    }

    return null;
}
```

These methods implement the *interface*, **SaveOpenInterface** which defines the protocol used by **Exercises** to make the application run more smoothly since relationships between classes aren't forced.⁹

```
// interface, that is used to save and open text documents
public interface SaveOpenInterface {

    public void SaveAs(String s, File f);

    public String Open(File f) throws IOException;

}
```

Similar to the **Open** method, the **UArmClientController** reads a file in the *current directory* to improve the functionality as an error wouldn't occur. (explained later)

```
@FXML
private JFXListView<Step> lvSteps;

ObservableList<Step> stepList = FXCollections.observableArrayList();

// initializes the UArmClientController, by adding all the steps executed
// by the application to the list
@FXML
private void initialize() throws SerialPortException, IOException {

    FilesUtil.fileExist("StepsList.txt");

    String content = FilesUtil.readTextFile("StepsList.txt");
    if (content.length() > 0) {
        String[] arr = content.split(";");
        stepList.clear();
    }
}
```

⁹ The Java™ Tutorial, editor. "What Is an Interface?" *IIT Kanpur*, www.iitk.ac.in/esc101/05Aug/tutorial/java/concepts/interface.html#:~:text=You%20use%20an%20interface%20to,artificially%20forcing%20a%20class%20relationship. Accessed 15 Jan. 2021.

```

        for (int i = 0; i < arr.length; i++) {
            String[] arr2 = arr[i].split(":");
            stepList.add(new Step(arr2[0], Integer.parseInt(arr2[1])));
        }

        this.lvSteps.setItems(this.stepList);
    }
}

```

When working with files, the following methods are necessary to read and write files in the *current directory*. Developed using the **Files** class to provide a system-independent view of paths and simple manipulation of user accessible files.¹⁰

```

public class FilesUtil {

    // Reads the text document as a string
    public static String readTextFile(String fileName) throws IOException {
        String content = new String(Files.readAllBytes(Paths.get(fileName)));
        return content;
    }

    // Reads the text document as a string, but by lines
    public static List<String> readTextFileByLines(String fileName) throws IOException {
        List<String> lines = Files.readAllLines(Paths.get(fileName));
        return lines;
    }

    // Writes a String into the text document, and saves it at the directory
    public static void writeToTextFile(String fileName, String content) throws IOException {
        Files.deleteIfExists(Paths.get(fileName));
        Files.write(Paths.get(fileName), content.getBytes(), StandardOpenOption.CREATE);
    }
}

```

The following method uses an *append* to logs all steps executed.

```

// Adds a new String in an existing file
public static void writeOverTextFile(String fileName, String content) throws IOException {
    content = content + "\r\n\r\n";
    Files.write(Paths.get(fileName), content.getBytes(), StandardOpenOption.APPEND);
}

```

The following method checks if the file being read is in the *current directory*, if not, it's created to ensure there aren't errors when reading files. This improves flexibility as it creates a relative path for the application, making it work in any location installed.

```

// Checks if the file exists, if not, then create the file
public static void fileExist(String fileName) throws IOException {
    Path path = Paths.get(fileName);
    boolean pathExists = Files.exists(path);
    if (!pathExists) {
        FilesUtil.writeToTextFile(fileName, "");
    }
}
}

```

¹⁰ "Class File." Oracle, docs.oracle.com/javase/7/docs/api/java/io/File.html. Accessed 13 Jan. 2021.

The **ImExportExerciseController** contains a list of exercises in three levels (easy, medium, hard).

Figure 8- *ImExportExerciseController*

Import or Export Exercises

Back

Importing/Exporting Exercises

Add Exercise

Easy ▼

+

Export Import

Easy: Move the Arm right and then left, back to its original position.

Medium: Put the item anywhere the arm can reach and try to grab the ball with the Arm.

Hard: Try to find the limitations of this Arm.

Edit Delete

Clear

To improve organization, the following method automatically sorts the list of exercise when exercises are edited or added.

```
// Sorts the exercises from Easy, Medium to Hard
@FXML
void sort() throws IOException {
    save();
    ReadExercises method = new ReadExercises();
    String content = FilesUtil.readFile("Exercises.txt");
    // Creates a Queue from giving a string that will be sorted in the
    // ReadExercises class located in Methods package then adds the
    // items one by one, thereby sorting the list
    if (content != null) {
        Queue<String> exercises = method.sortExercise(content);
        exerciseList.clear();
        while (exercises.size() != 0) {
            String type = exercises.remove();
            String ex = exercises.remove();
            exerciseList.add(new Exercise(type, ex));
        }
        save();
    }
}
```

This invokes a method called “sortExercise” located in the **ReadExercises** class, which utilizes a *Queue to sort exercises by level*. The method uses the *enqueue* and *dequeue* concepts (**add()** and **remove()** in Java) and it outmatches arrays and stacks.

```
// Sorts the list of exercises from Easy, Medium to Hard
public Queue<String> sortExercise(String a) {
    Queue<String> exercises = new LinkedList<>();
    String[] exercise = a.split(";");
    for (int i = 0; i < 3; i++) {
        String type = null;
        if (i == 0)
            type = "Easy";
        if (i == 1)
            type = "Medium";
        if (i == 2)
            type = "Hard";
        for (int j = 0; j < exercise.length; j++) {
            String[] temp = exercise[j].split(":");
            if (temp[0].equals(type)) {
                exercises.add(temp[0]);
                exercises.add(temp[1]);
            }
        }
    }
    return exercises;
}
```

The two loops iterate three times for each level and *enqueues* them to the list.

The **ReadExercises** class contains three *private Array Lists* for each level since there isn't a set size, so there isn't a limit to the number of exercises to being added. Moreover, searching isn't necessary since its already organized in levels.

```
//this class contains methods regarding the exercises
public class ReadExercises {
    private List<String> EExercise = new ArrayList<>();
    private List<String> MExercise = new ArrayList<>();
    private List<String> HExercise = new ArrayList<>();
    private static int level = 0;
    private static int quesNum = 0;

    // set level for difficulty
    public static void setLevel(int num) {
        level = num;
    }

    // get level for difficulty
    public static int getLevel() {
        return level;
    }

    // get number of question
    public static int getQuesNum() {
        return quesNum;
    }

    // increase the number of the question
    public static void increaseQuesNum() {
        quesNum++;
    }
}
```



```
// decrease the number of the question
public static void decreaseQuesNum() {
    quesNum--;
}

// reset the number of the question
public static void resetQuesNum() {
    quesNum = 0;
}
```

The following method (one out of three) returns the exercise at the chosen level.

```
// Gets all the exercises at a Easy level
public String getEasyExercises() throws IOException {
    String content = FilesUtil.readTextFile("Exercises.txt");
    String[] arr = content.split(";");
    if (arr.length == 0)
        return "No Exercise Available";

    getExercise(arr, "Easy", EExercise);

// When no more exercises at available, then it outputs "No Exercise Available"
    if (quesNum >= EExercise.size()) {
        quesNum = EExercise.size();
        return "No Exercise Available";
    }
    return "Easy:" + EExercise.get(quesNum);
}
```

For user-friendliness, a *Breakpoint* is used, to inform when there aren't any more exercises.

To get rid of redundancies, instead of creating different connection controllers, an *if statement* is used.

Moreover, if the **uArm** isn't connected, an error appears since it's not possible to use it.

```
// Try to connect with the UArm, calling a method in Uarm.java and using public variables
@FXML
void btConnect_clicked(ActionEvent event) throws SerialPortException, IOException {
// Checks if the user has chosen the port that should be connected with the Arm, if not,
// then an error would appear
    if (getPort() == null) {
        BorderPane root = (BorderPane) FXMLLoader.load(getClass().getResource("Error.fxml"));
        Scene scene = new Scene(root, 525, 155);
        scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());

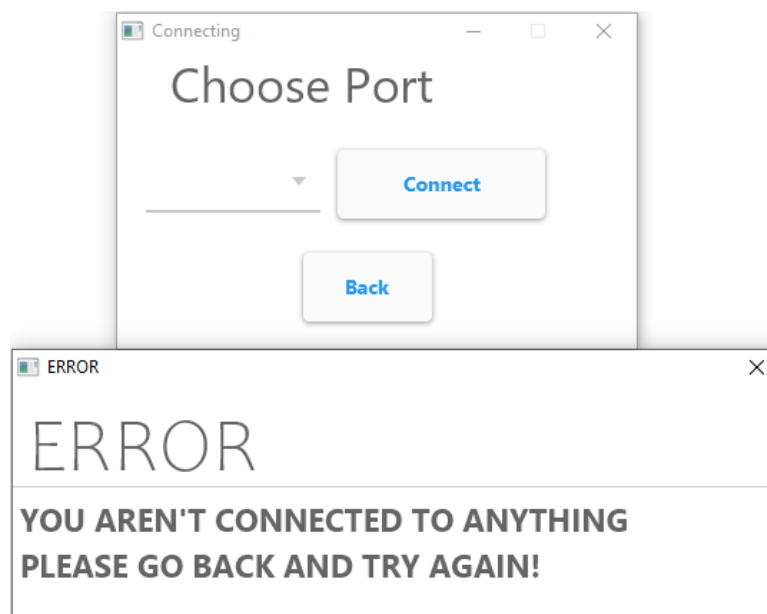
        Stage primaryStage = new Stage();
        primaryStage.setScene(scene);
        primaryStage.setTitle("ERROR");
        primaryStage.initModality(Modality.APPLICATION_MODAL);
        primaryStage.setResizable(false);
        primaryStage.setFullScreen(false);
        primaryStage.show();
    } else {
        ChosenPort = getPort();
        uarm = new UArm(ChosenPort);
        uarm.setPort();
        String gui = "UarmClient.fxml";
        String title = "uArm Client";
    }
}
```



```
// Looks at if the exercise level already stated is at which level and then opens the
// completing exercises window or the uArm Client
int level = ReadExercises.getLevel();
if (level == 1 || level == 2 || level == 3) {
    gui = "UarmClientExercise.fxml";
    title = "Completing Exercises";
}
Parent tableViewParent = FXMLLoader.load(getClass().getResource(gui));
Scene tableViewScene = new Scene(tableViewParent);
Stage window = (Stage) ((Node) event.getSource()).getScene().getWindow();

window.setScene(tableViewScene);
window.centerOnScreen();
window.setTitle(title);
window.show();
}
}
```

Figure 9- Connecting to uArm



```
// Initializes the ChoosePortController by getting all the ports
// that are connected to the computer, the uArm
@FXML
private void initialize() throws SerialPortException {

    SerialPort[] portNames = SerialPort.getCommPorts();
    for (int i = 0; i < portNames.length; i++)
        this.portList.add(portNames[i].getSystemPortName());

    ports.setItems(portList);
}
```

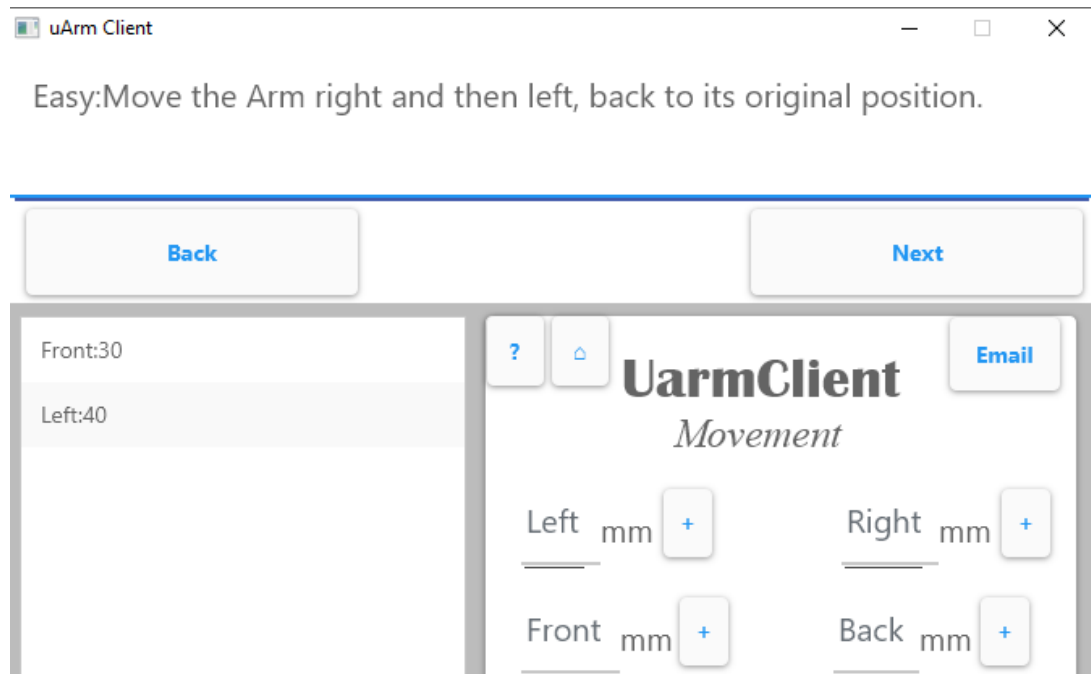
This shows the connections by using the *jSerialComm* library as its superior when getting the name of connected ports as it doesn't require external libraries.¹¹

```
import com.fazecast.jSerialComm.SerialPort;
```

¹¹ Fazecast, editor. "jSerialComm." *GitHub*, [fazecast.github.io/jSerialComm/](https://github.com/fazecast/jSerialComm/). Accessed 6 Jan. 2021.

The following GUI is to complete exercises.

Figure 10- Completing Exercises



Initializing the controller to get the exercise corresponding to level.

```
//this class is to complete exercises with the arm, using the
//UArmClientController as its base
public class UArmClientExerciseController extends UArmClientController {

    @FXML
    private VBox bpClient;

    // Initializing the controller to complete exercises in, which would
    // present the exercise at the level chosen
    @FXML
    private void initialize() throws IOException, SerialPortException {

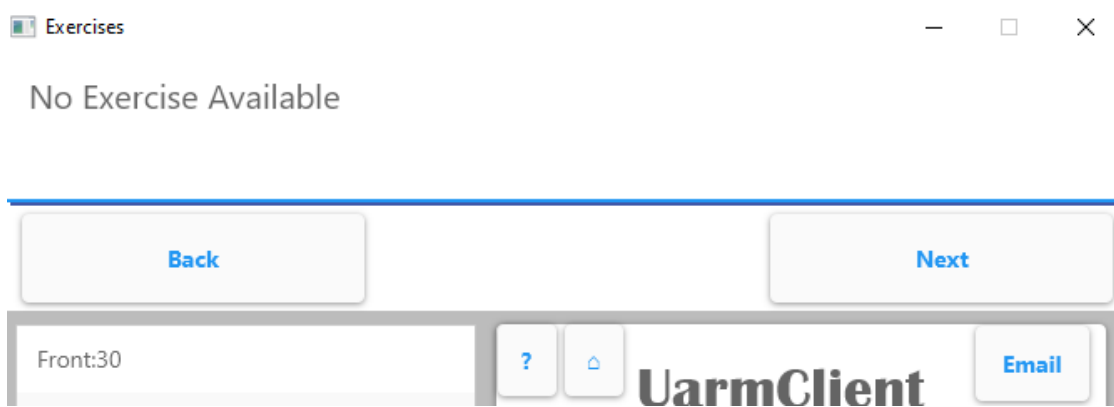
        FilesUtil.fileExist("Exercises.txt");

        ReadExercises e = new ReadExercises();

        taExercise.setFont(new Font(20));
        // checks the level of the exercises chosen and gets the level of exercises
        int level = ReadExercises.getLevel();
        if (level == 1) {
            taExercise.setText(e.getEasyExercises());
        } else if (level == 2) {
            taExercise.setText(e.getMediumExercises());
        } else if (level == 3) {
            taExercise.setText(e.getHardExercises());
        }
        // Adds the UarmClient window below the exercises
        BorderPane temp = FXMLLoader.load(getClass().getResource("UarmClient.fxml"));
        bpClient.getChildren().add(temp);
    }
}
```

Inheritance is used to reduce code when duplicating the **UArmclientController's** functionalities and GUI.

Figure 11- No Exercise Available



The **Back** button boosts navigation allowing the user to select the level again or see the previous exercise.

```
// Goes back to the previous window
@FXML
void btBack_Clicked(ActionEvent event) throws IOException, SerialPortException {
// Reduces the question number, to go back to the previous question or previous window
ReadExercises.decreaseQuesNum();
if (ReadExercises.getQuesNum() == -1) {
    ReadExercises.setLevel(0);
    ReadExercises.resetQuesNum();
    Parent tableViewParent = FXMLLoader.load(getClass().getResource("ExerciseChooser.fxml"));
    Scene tableViewScene = new Scene(tableViewParent);

    Stage window = (Stage) ((Node) event.getSource()).getScene().getWindow();
    window.setScene(tableViewScene);
    window.setTitle("Choose Hardness");
    window.centerOnScreen();
    window.show();
    uarm.serialPort.closePort();
} else {
    Parent tableViewParent = FXMLLoader.load(getClass().getResource("UarmClientExercise.fxml"));
    Scene tableViewScene = new Scene(tableViewParent);

    Stage window = (Stage) ((Node) event.getSource()).getScene().getWindow();
    window.setScene(tableViewScene);
    window.setTitle("Completing Exercises");
    window.centerOnScreen();
    window.show();
}
}
```

Furthermore, to improve efficiency, no matter how many times **Next** is pressed, pressing **Back** opens the last exercise (located in the method to get exercises).

```
// When no more exercises at available, then it outputs "No Exercise Available"
if (quesNum >= EExercise.size()) {
    quesNum = EExercise.size();
    return "No Exercise Available";
}
```

To enhance the application's service, the files/progress can be sent through email. The **Email** class is in the **Methods** package, an example of *modular programming* used to make extensibility easier, improve the management and maintenance of the application.

```
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

public class Email {
    public static void send(String receiver, String subject) throws IOException {
        String to = receiver;
        String from = "uarmclient@gmail.com";
        String host = "smtp.gmail.com";
        String port = "465";
        String emailName = "From the UarmClient: " + subject;

        Properties prop = new Properties();
        prop.put("mail.smtp.host", host);
        prop.put("mail.smtp.port", port);
        prop.put("mail.smtp.auth", "true");
        prop.put("mail.smtp.socketFactory.port", port);
        prop.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

        final String username = from;
        final String password = [REDACTED];

        Session session = Session.getInstance(prop, new javax.mail.Authenticator() {
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(username, password);
            }
        });
    }
}
```

The *outgoing mail server*, smtp.gmail.com is used because it's free and easily sends emails with attachments as it uses a two-step verification, Gmail account, host, port and its properties, internet connection.¹²

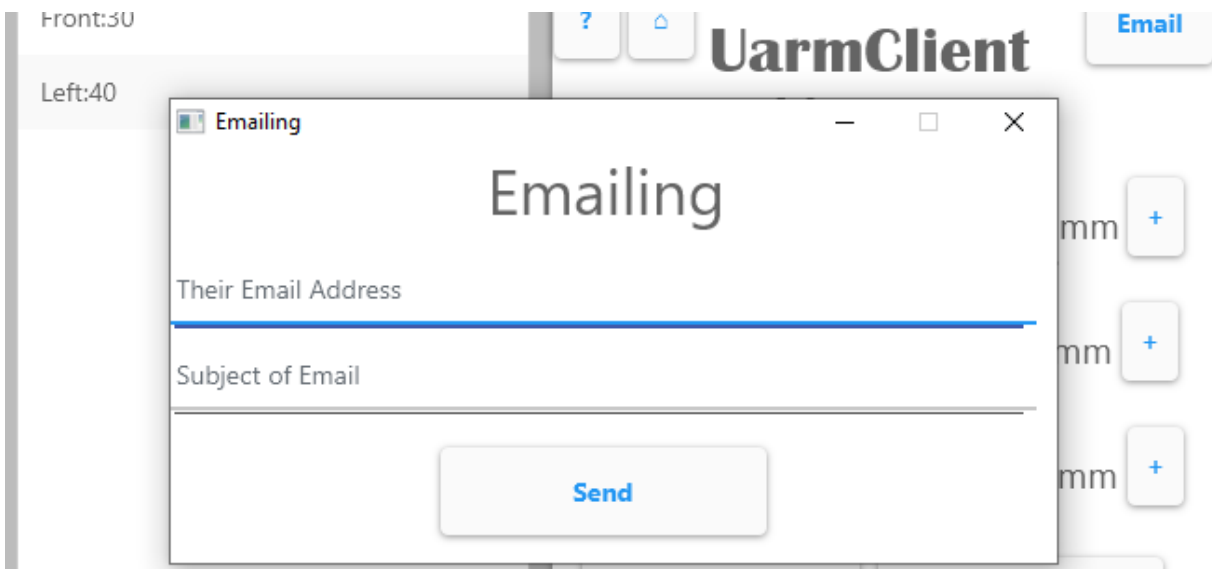
¹² SiteGround, editor. "Google SMTP server - how to send emails for free?" *SiteGround*, [www.siteground.com/kb/google_free_smtp_server/#:~:text=Google's%20Gmail%20SMTP%20server%20is,can%20use%20to%20send%20emails.&text=Outgoing%20Mail%20\(SMTP\)%20Server%3A,mail%20client%2Fwebsite%20SMTP%20plugin](https://www.siteground.com/kb/google_free_smtp_server/#:~:text=Google's%20Gmail%20SMTP%20server%20is,can%20use%20to%20send%20emails.&text=Outgoing%20Mail%20(SMTP)%20Server%3A,mail%20client%2Fwebsite%20SMTP%20plugin). Accessed 17 Jan. 2021.

```

try {
    Message message = new MimeMessage(session);
    message.setFrom(new InternetAddress(from));
    message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(to));
    message.setSubject(emailName);
    Multipart emailContent = new MimeMultipart();
    MimeBodyPart body = new MimeBodyPart();
    body.attachFile("StepsList.txt");
    MimeBodyPart body1 = new MimeBodyPart();
    body1.attachFile("Exercises.txt");
    MimeBodyPart body2 = new MimeBodyPart();
    body2.attachFile("Log.txt");
    emailContent.addBodyPart(body);
    emailContent.addBodyPart(body1);
    emailContent.addBodyPart(body2);
    message.setContent(emailContent);
    Transport.send(message);
} catch (MessagingException mex) {
    mex.printStackTrace();
}
}

```

Figure 12- Email



As shown in this Criterion, most of the methods implemented were ideas thought of from the planning stage. Even though there were some changes, the concepts used were planned out as they helped me more efficiently create the application. Additionally, these ideas, preparations, developments, and implementations were effective and utilized thanks to learning computer science in the last two years of high school.

Word Count: 1065 (excluding Titles, Captions and Parentheses)

List of References

- "Class BufferedReader." *Oracle*, docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html. Accessed 10 Jan. 2021.
- "Class Date." *Oracle*, docs.oracle.com/javase/7/docs/api/java/util/Date.html. Accessed 8 Jan. 2021.
- "Class File." *Oracle*, docs.oracle.com/javase/7/docs/api/java/io/File.html. Accessed 13 Jan. 2021.
- "Class FileChooser." *Oracle*, docs.oracle.com/javase/8/javafx/api/javafx/stage/FileChooser.html. Accessed 6 Jan. 2021.
- "Class PrintStream." *Oracle*, docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html. Accessed 15 Jan. 2021.
- "Class SimpleDateFormat." *Oracle*, docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html. Accessed 6 Jan. 2020.
- "Enum TimeUnit." *Oracle*, docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html. Accessed 8 Jan. 2021.
- Fazecast, editor. "jSerialComm." *GitHub*, fazecast.github.io/jSerialComm/. Accessed 6 Jan. 2021.
- The Java™ Tutorial, editor. "What Is an Interface?" *IIT Kanpur*,
www.iitk.ac.in/esc101/05Aug/tutorial/java/concepts/interface.html#:~:text=You%20use%20an%20inter
face%20to,artificially%20forcing%20a%20class%20relationship. Accessed 15 Jan. 2021.
- "jfoenix documentation." *JFoenix*, www.jfoenix.com/documentation.html. Accessed 10 Jan. 2021.
- "jSSC (Java Simple Serial Connector)." *javalibs*, javalibs.com/artifact/org.scream3r/jssc. Accessed 8 Jan. 2021.
- SiteGround, editor. "Google SMTP server - how to send emails for free?" *SiteGround*,
www.siteground.com/kb/google_free_smtp_server/#:~:text=Google's%20Gmail%20SMTP%20server%
20is,can%20use%20to%20send%20emails.&text=Outgoing%20Mail%20(SMTP)%20Server%3A,mail
%20client%2Fwebsite%20SMTP%20plugin). Accessed 17 Jan. 2021.