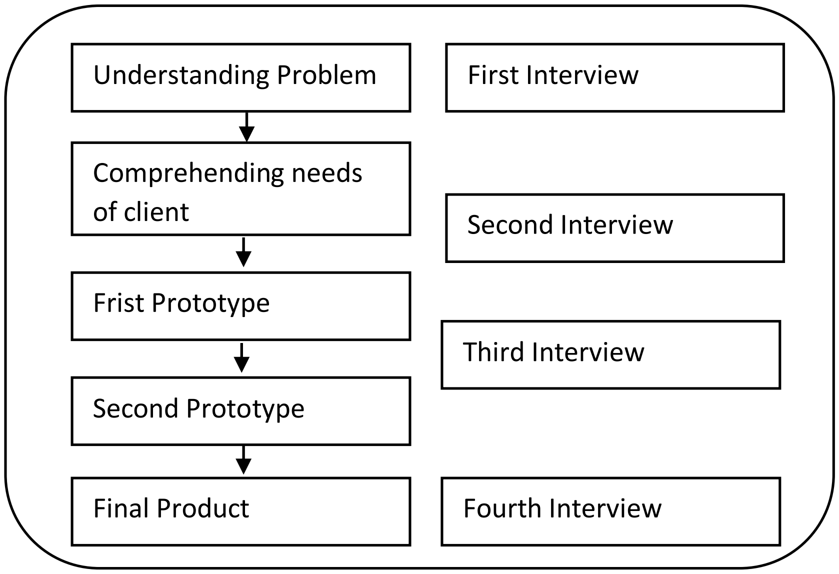
Criterion B: Design

Development Sequence

Figure 1- Overview of Development

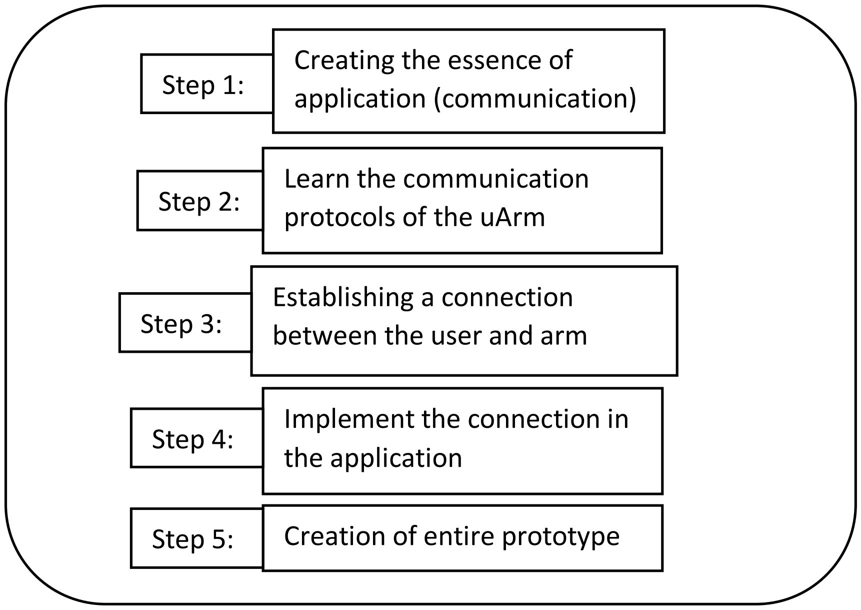


Constructing this application will require the cooperation of the client (interviews) when producing the final application, to accurately fulfill the needs of the client.

Planning of First Prototype

To construct the first prototype the following steps are necessary.

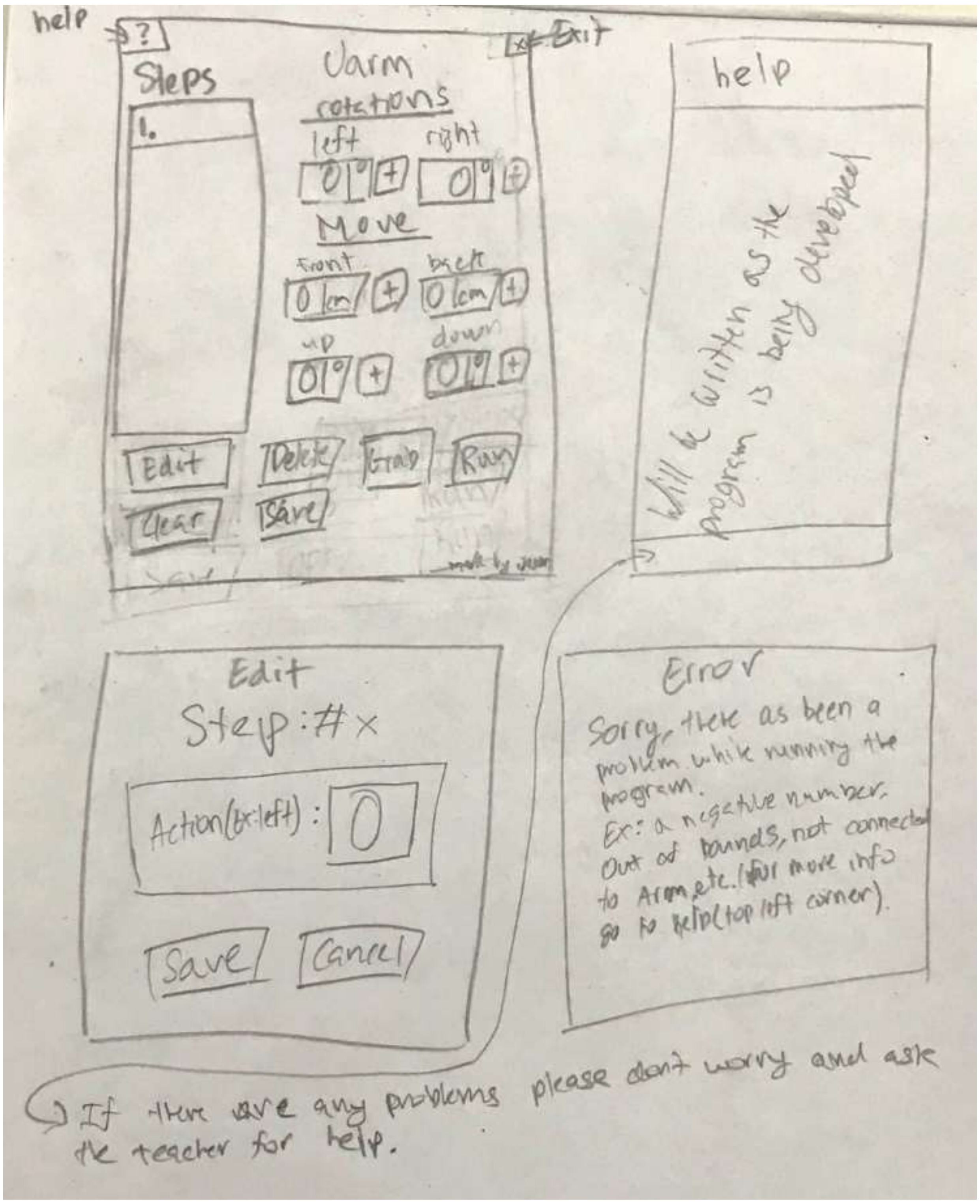
Figure 2- Steps of Development


Step 1: Creating the essence of application (communication)
Step 2: Learn the communication protocols of the uArm
Step 3: Establishing a connection between the user and arm
Step 4: Implement the connection in the application
Step 5: Creation of entire prototype

Development of First Prototype

The following sketch was shown to and discussed with the client to understand his needs from the feedback given (see Appendix 2). Additionally, this will serve as the window to communicate with the arm, which is beneficial to me when testing the arm's limitations and functionality when developing the first prototype.

Figure 3- Essence of Prototype (drawing)



Sorry, there as been a problem while running the program.
Ex: a negative number, Out of bounds, not connected to Arm, etc. (for more info go to help(top left corner).

If there are any problems please don't worry and ask the teacher for help.

The following window like the others are the digital version of the design of the application (GUI) created with SceneBuilder.

Figure 4- GUI of first prototype (Main Form)

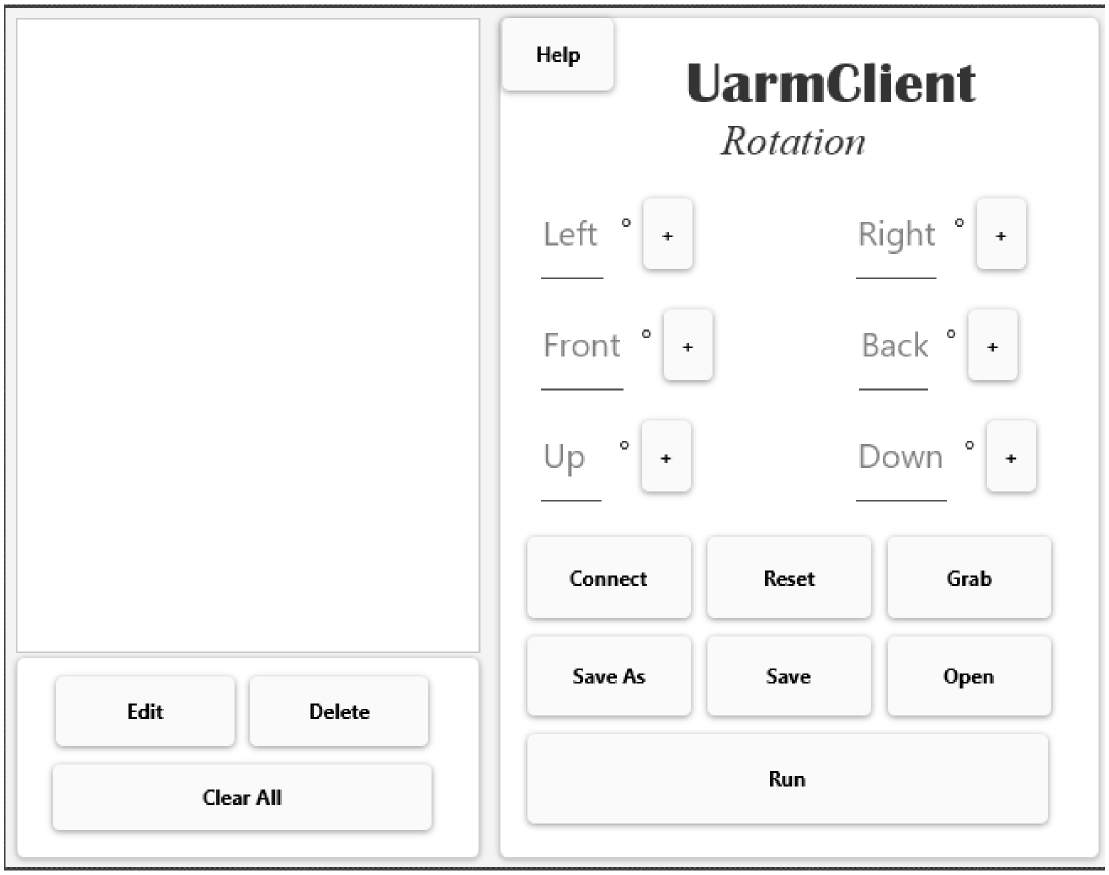
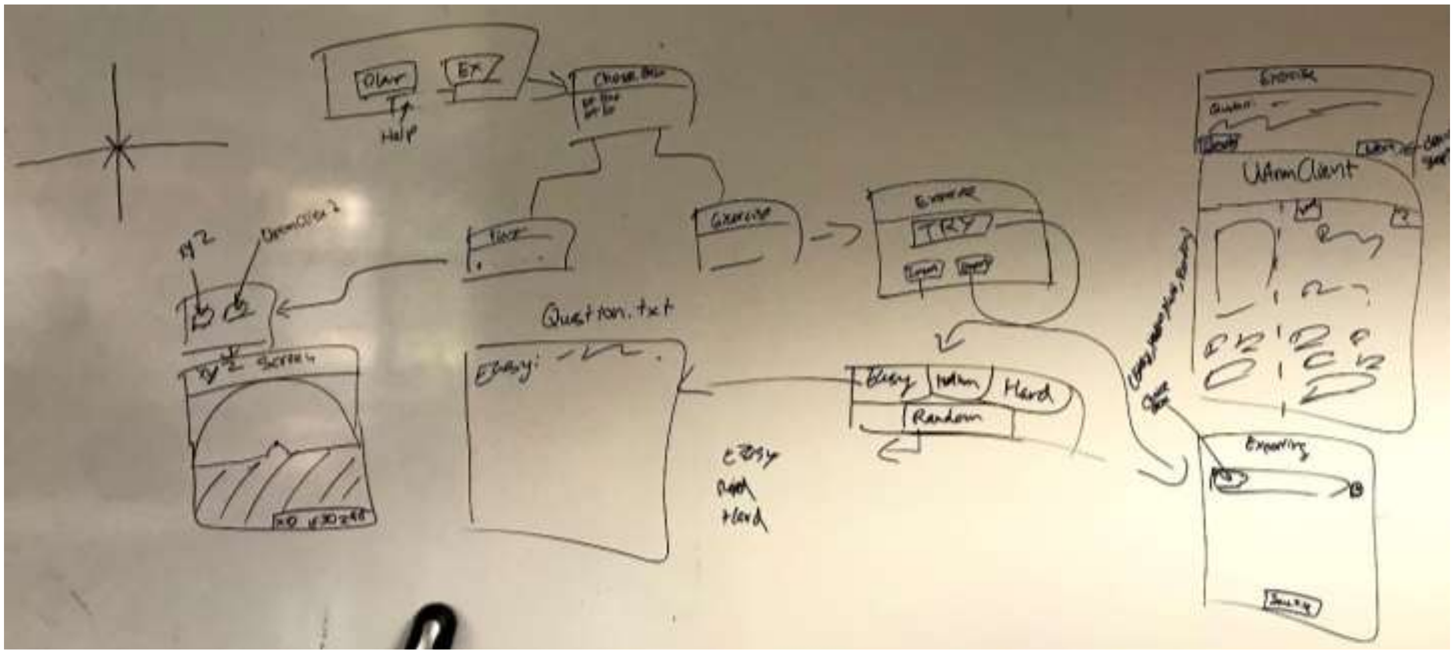


Figure 5- First prototype (Drawing)



This drawing is mainly used by me, to help illustrate the relationships of the classes, variables and the GUIs that I will implement in the first prototype.

Figure 6- Main Window

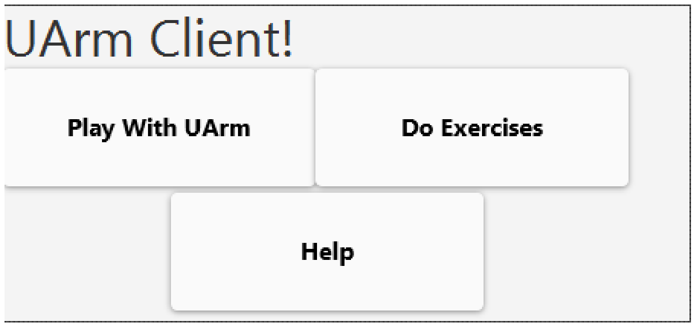
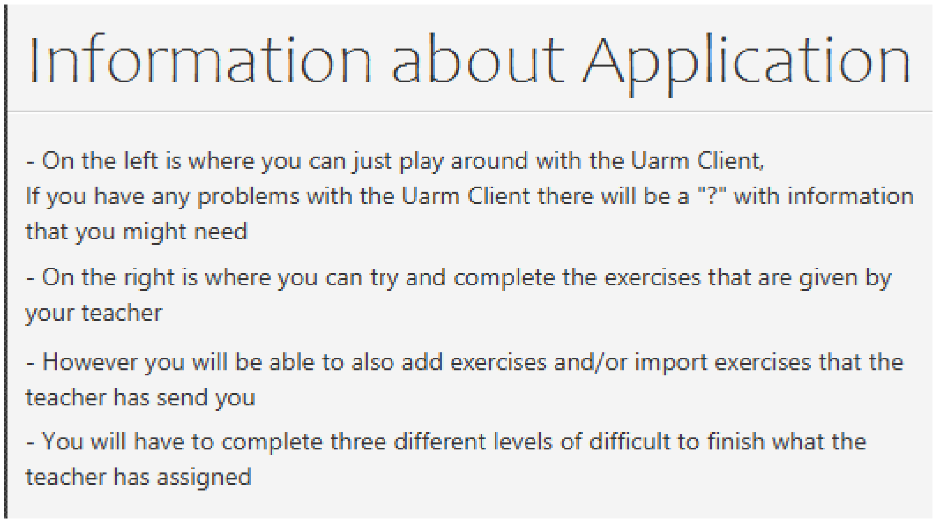


Figure 7- Overall information of application



Information about Application

- On the left is where you can just play around with the Uarm Client,
If you have any problems with the Uarm Client there will be a "?" with information that you might need

- On the right is where you can try and complete the exercises that are given by your teacher

- However you will be able to also add exercises and/or import exercises that the teacher has send you

- You will have to complete three different levels of difficult to finish what the teacher has assigned

Figure 8- Communication between user and arm

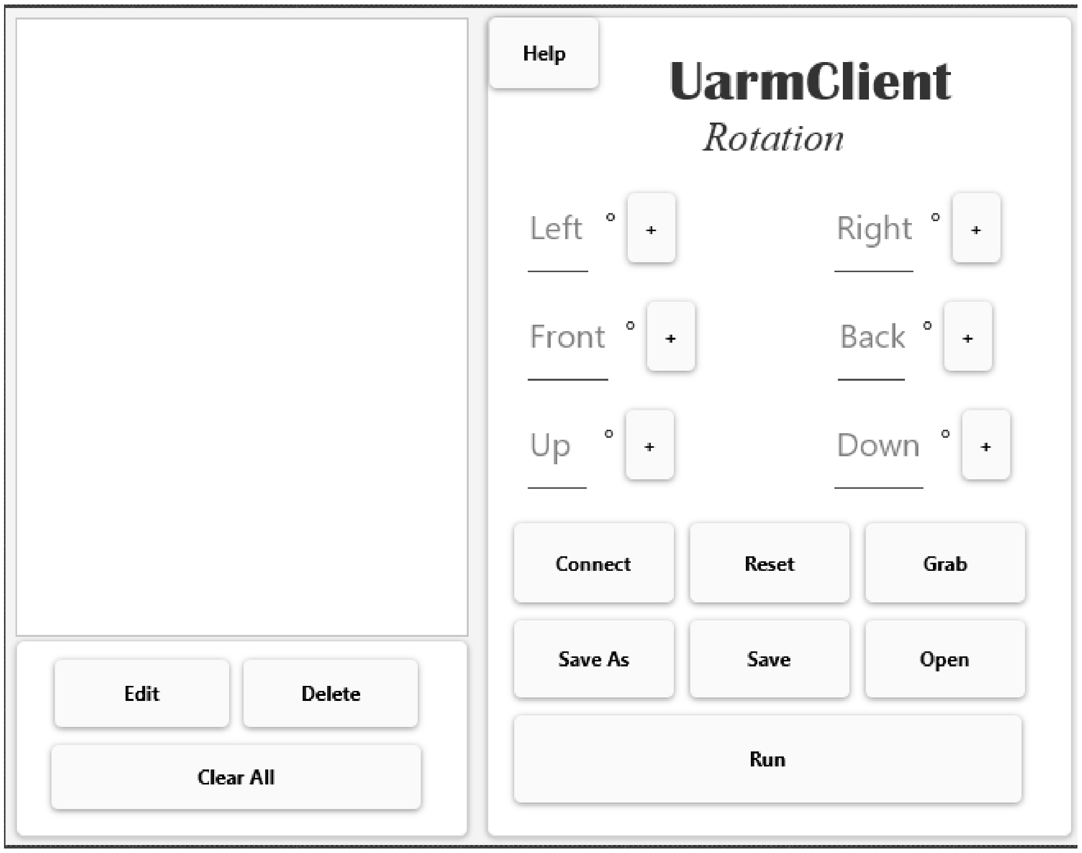


Figure 9- Edits steps

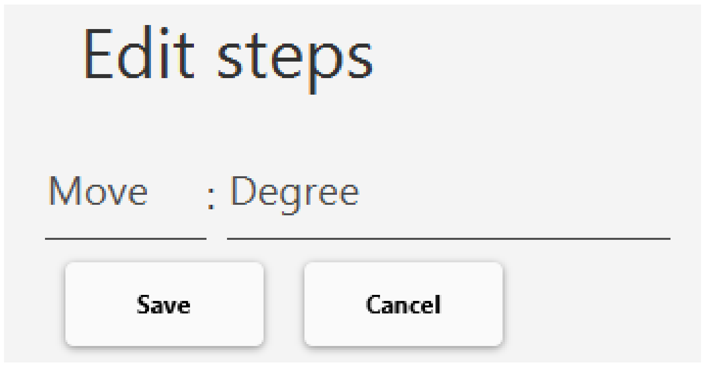
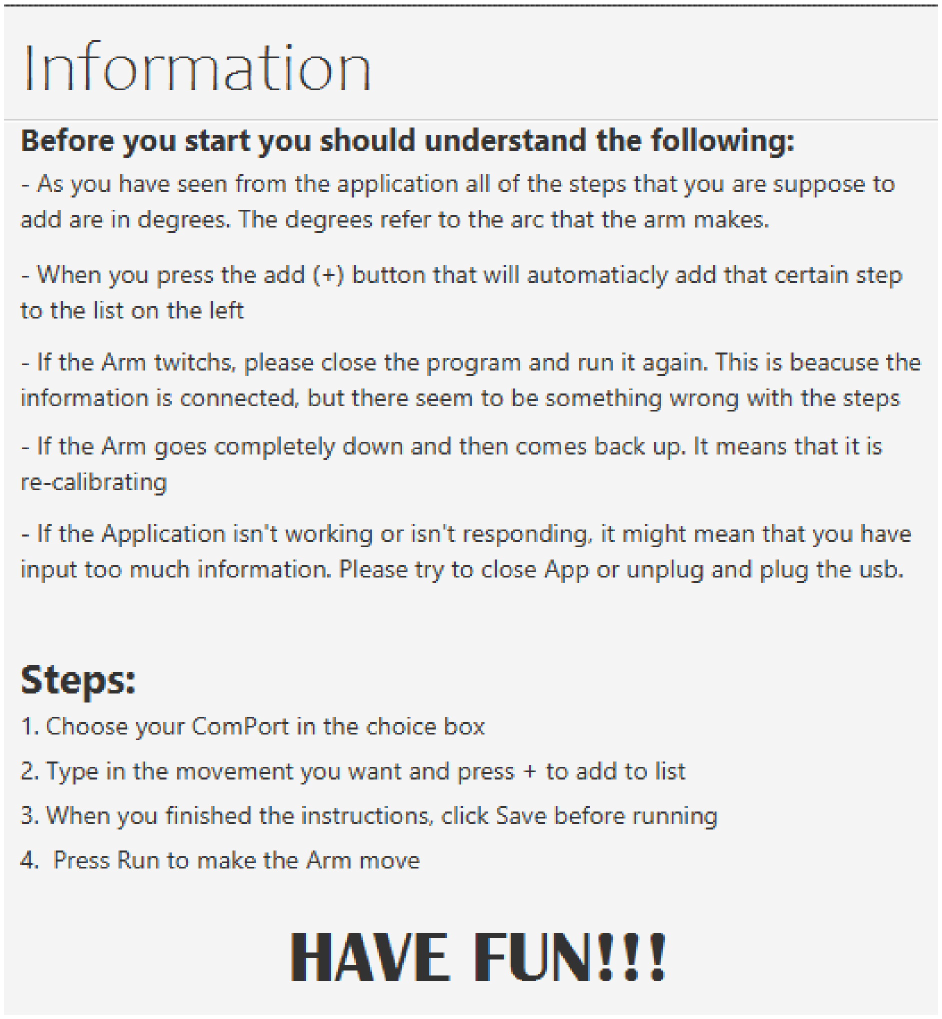


Figure 10- Information of communicating between user and arm



Information

Before you start you should understand the following:

- As you have seen from the application all of the steps that you are suppose to add are in degrees. The degrees refer to the arc that the arm makes.

- When you press the add (+) button that will automatiacly add that certain step to the list on the left

- If the Arm twitchs, please close the program and run it again. This is beacuse the information is connected, but there seem to be something wrong with the steps

- If the Arm goes completely down and then comes back up. It means that it is re-calibrating

- If the Application isn't working or isn't responding, it might mean that you have input too much information. Please try to close App or unplug and plug the usb.

Steps:

1. Choose your ComPort in the choice box
2. Type in the movement you want and press + to add to list
3. When you finished the instructions, click Save before running
4. Press Run to make the Arm move

HAVE FUN!!!

Figure 11- List of Exercises, that can be imported and exported

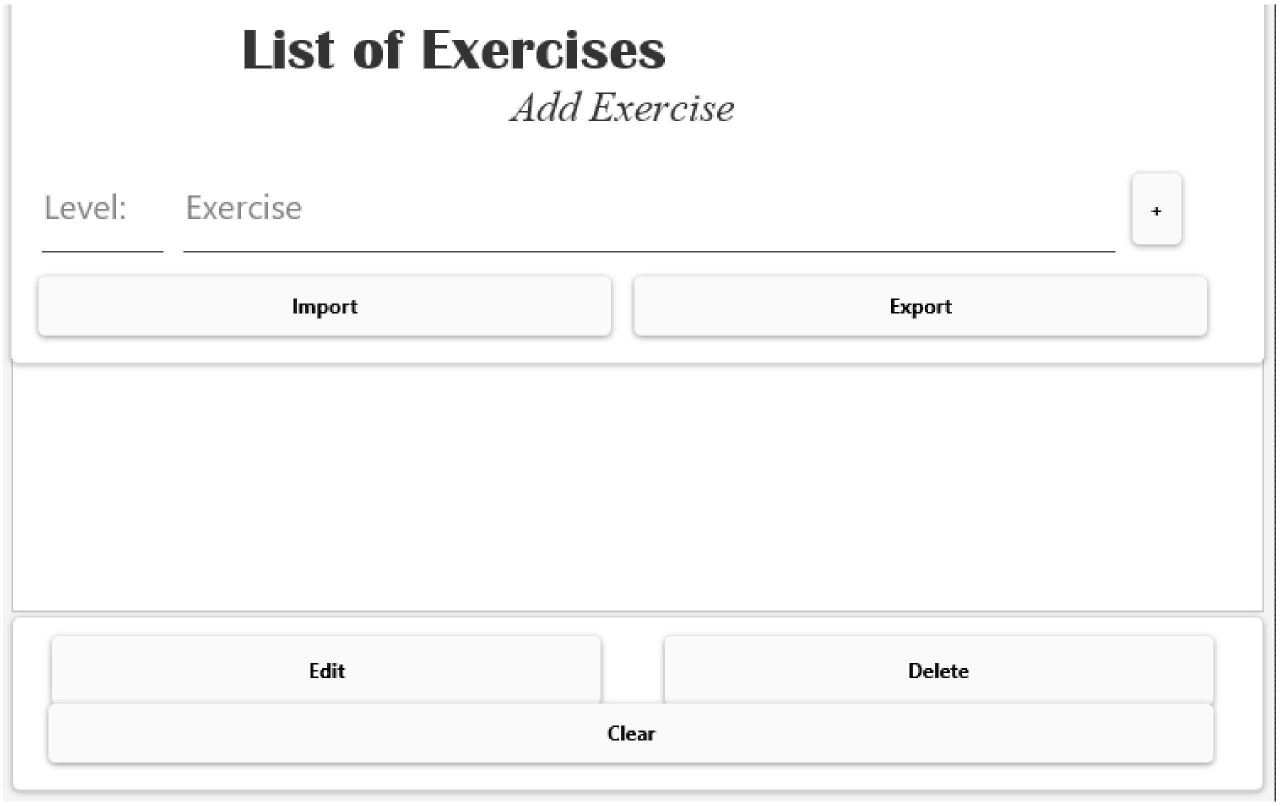


Figure 12- Edits exercises

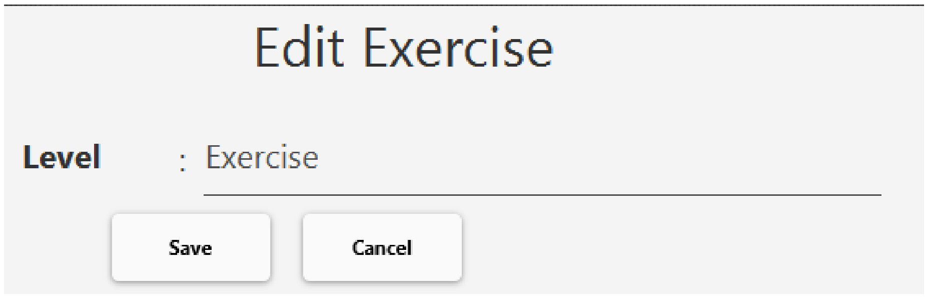
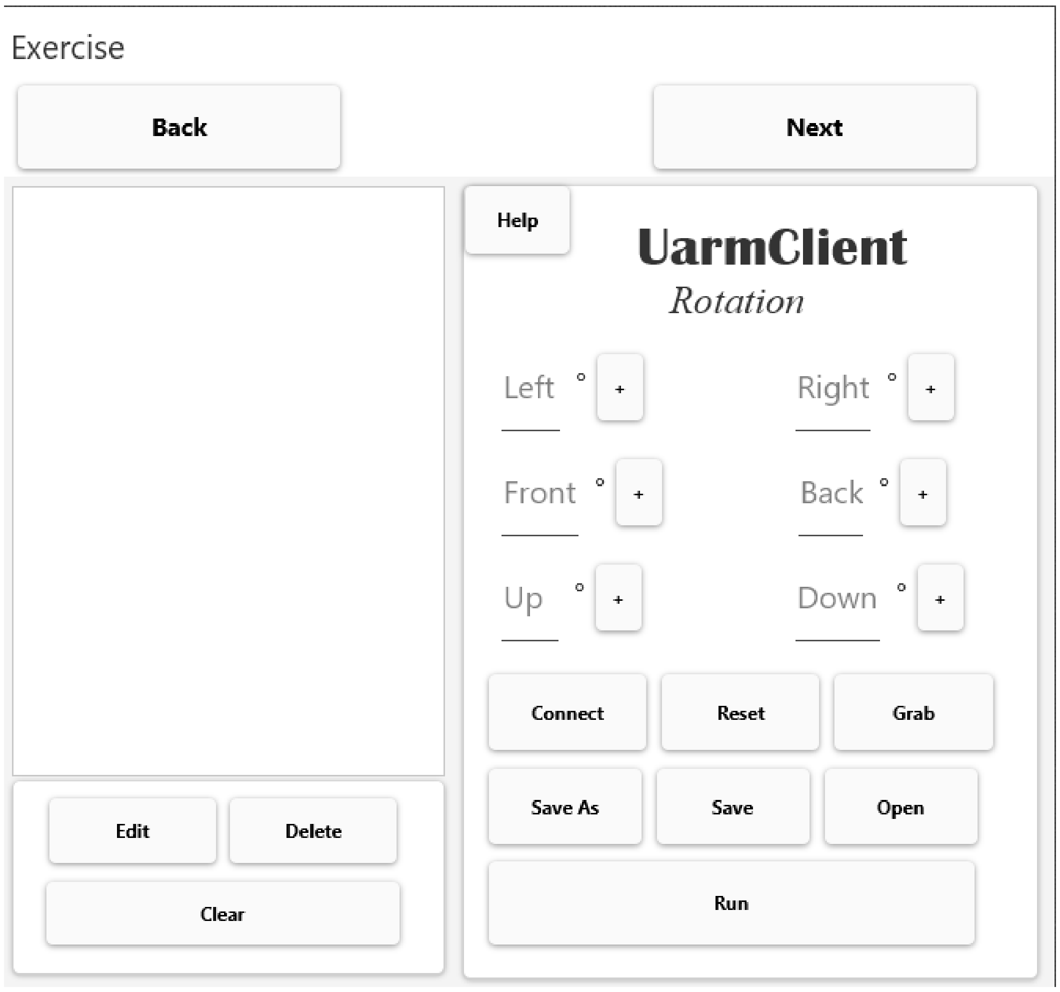


Figure 13- Completing exercises



The first prototype of the application would be shown to the client to get feedback (see Appendix 4).

Planning of Final Prototype

For the final prototype I will improve the overall aesthetic of the application by using JavaFx or JFoenix. Additionally, the following figures would be an improvement or addition of windows, developed from feedback given by client (see Appendix 4) and doesn't include windows that haven't been changed.

Figure 14- Main Window

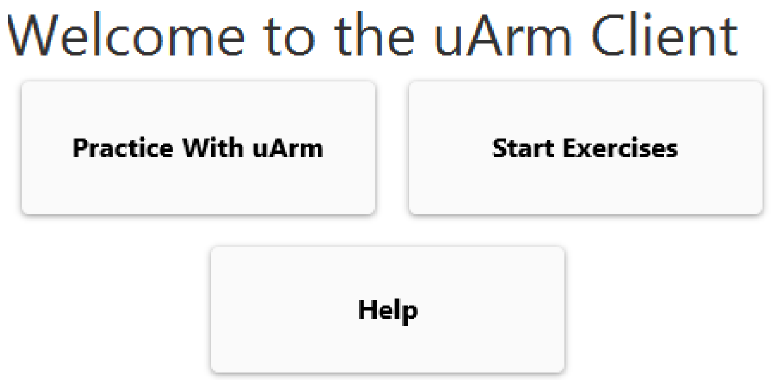


Figure 15- Establishing connection with user and arm

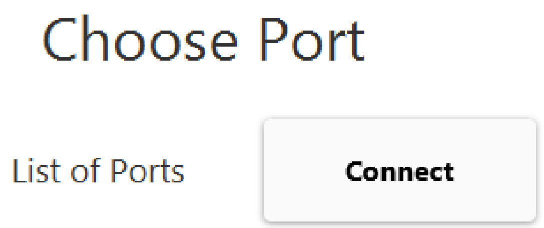


Figure 16- if connection isn't established between the user and the arm

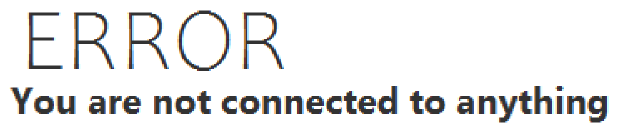


Figure 17- Communication between user and arm

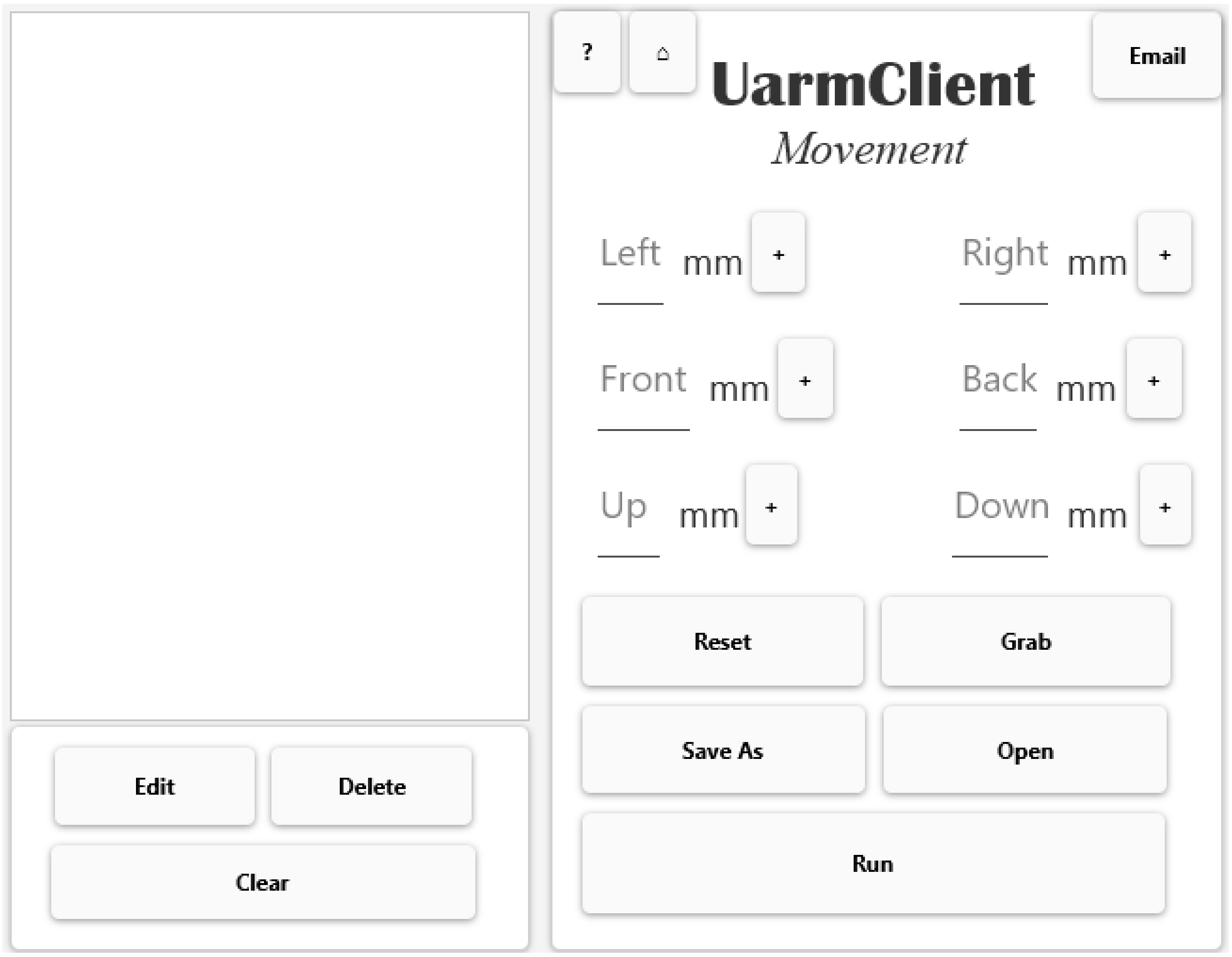


Figure 18- Edits steps

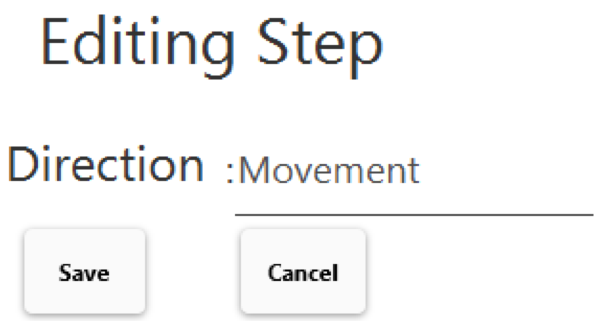
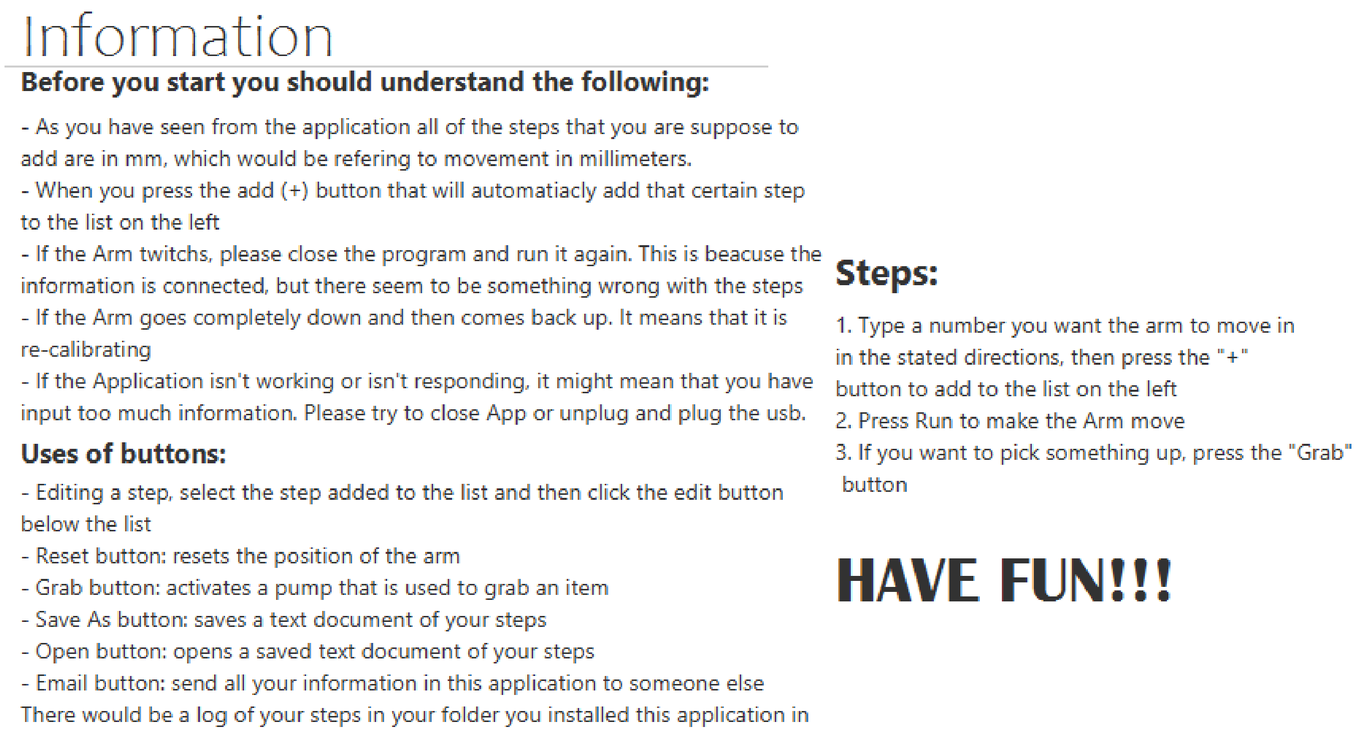


Figure 19- Information of communicating between user and arm


Information

Before you start you should understand the following:

- As you have seen from the application all of the steps that you are suppose to add are in mm, which would be refering to movement in millimeters.
- When you press the add (+) button that will automatiacly add that certain step to the list on the left
- If the Arm twitchs, please close the program and run it again. This is beacuse the information is connected, but there seem to be something wrong with the steps
- If the Arm goes completely down and then comes back up. It means that it is re-calibrating
- If the Application isn't working or isn't responding, it might mean that you have input too much information. Please try to close App or unplug and plug the usb.

Uses of buttons:

- Editing a step, select the step added to the list and then click the edit button below the list
- Reset button: resets the position of the arm
- Grab button: activates a pump that is used to grab an item
- Save As button: saves a text document of your steps
- Open button: opens a saved text document of your steps
- Email button: send all your information in this application to someone else

There would be a log of your steps in your folder you installed this application in

Steps:

1. Type a number you want the arm to move in in the stated directions, then press the "+" button to add to the list on the left
2. Press Run to make the Arm move
3. If you want to pick something up, press the "Grab" button

HAVE FUN!!!

Figure 20- Emailing all the information of the application

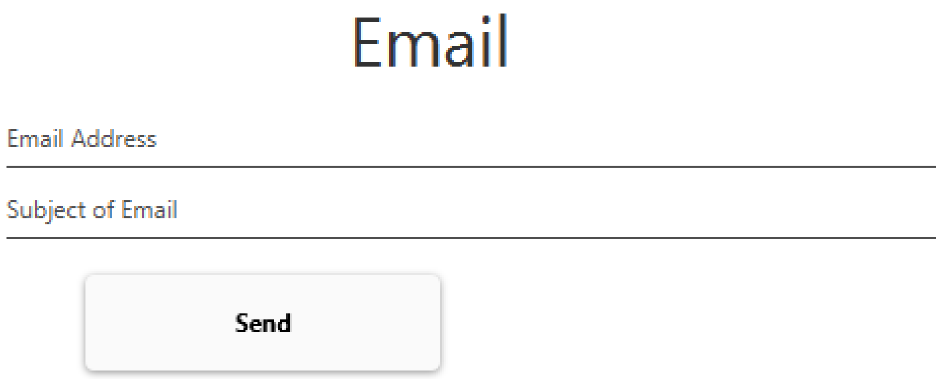


Figure 21- Choosing the level of difficulty for exercises

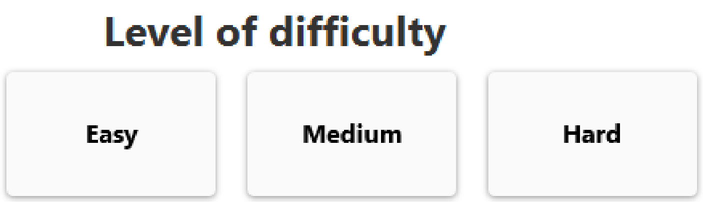


Figure 22- List of Exercises, that can be imported and exported

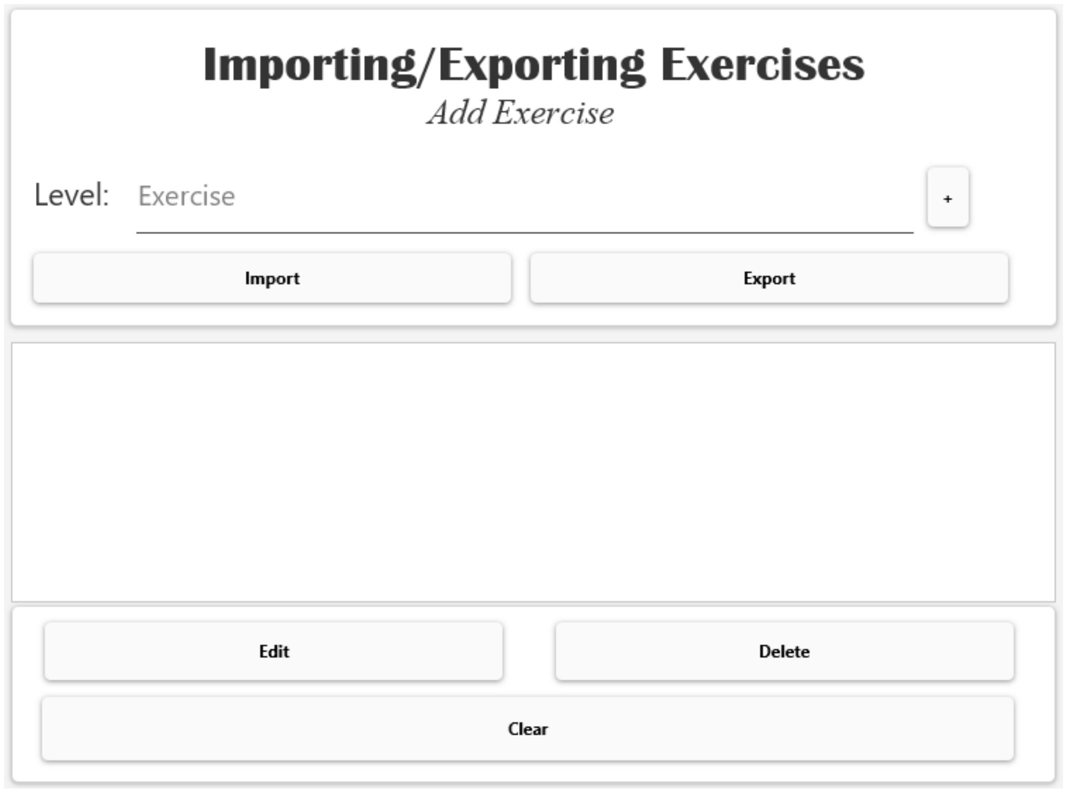
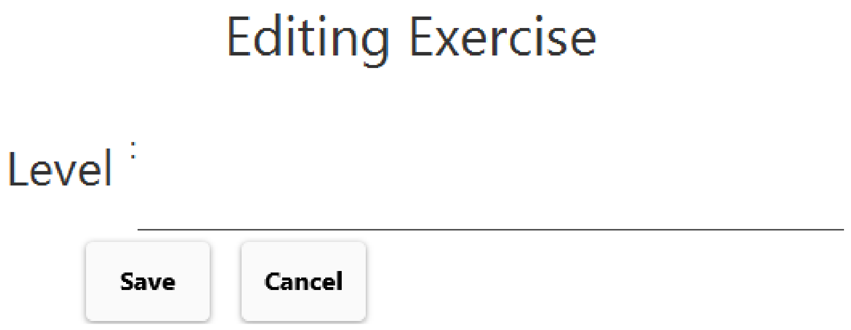


Figure 23- Edits exercises



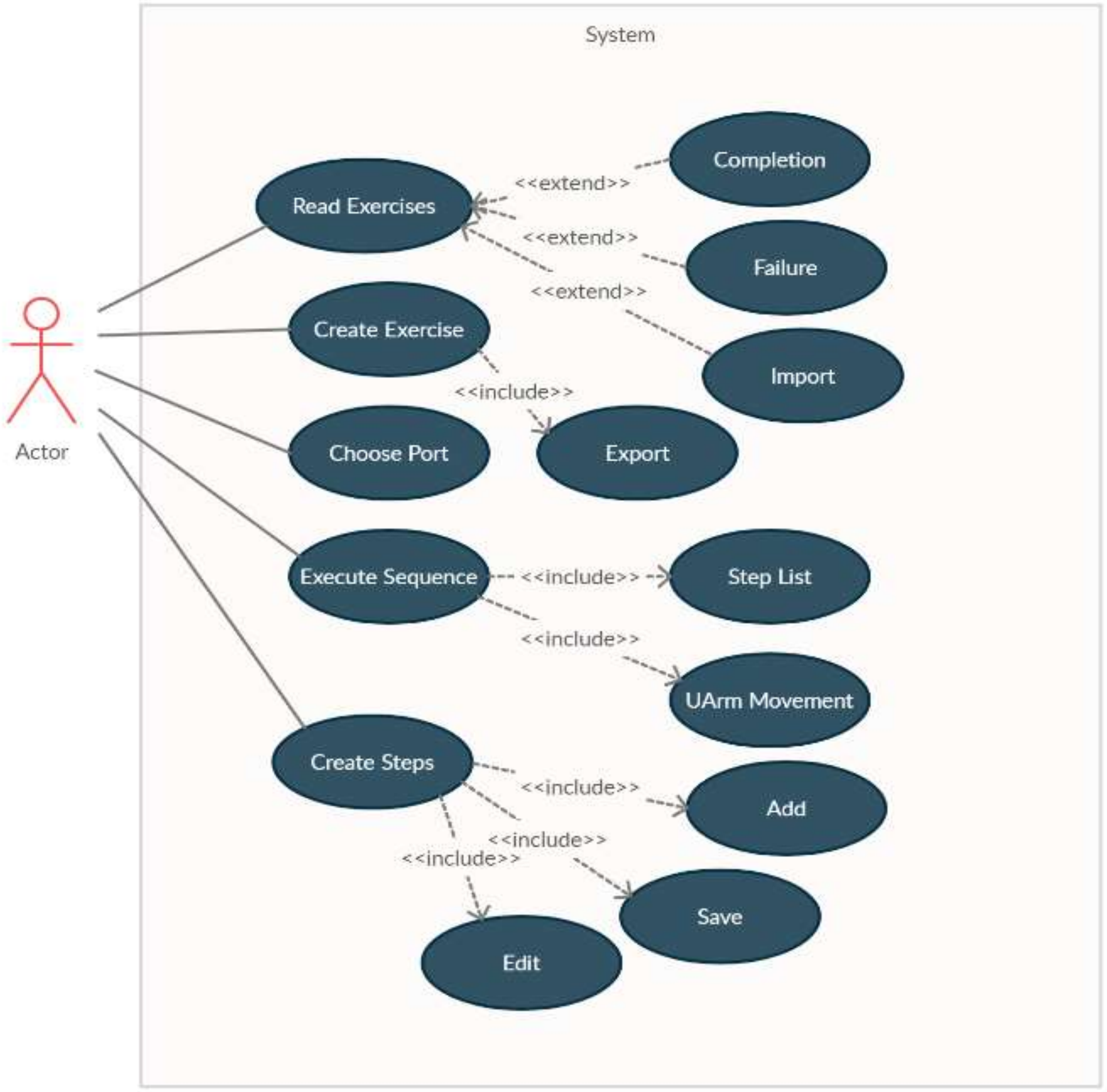
These GUI are created in this fashion because they look more aesthetically pleasing as well as being more simplistic, since middle schoolers will be using this application.

Domain Models

UML Use Case Diagrams

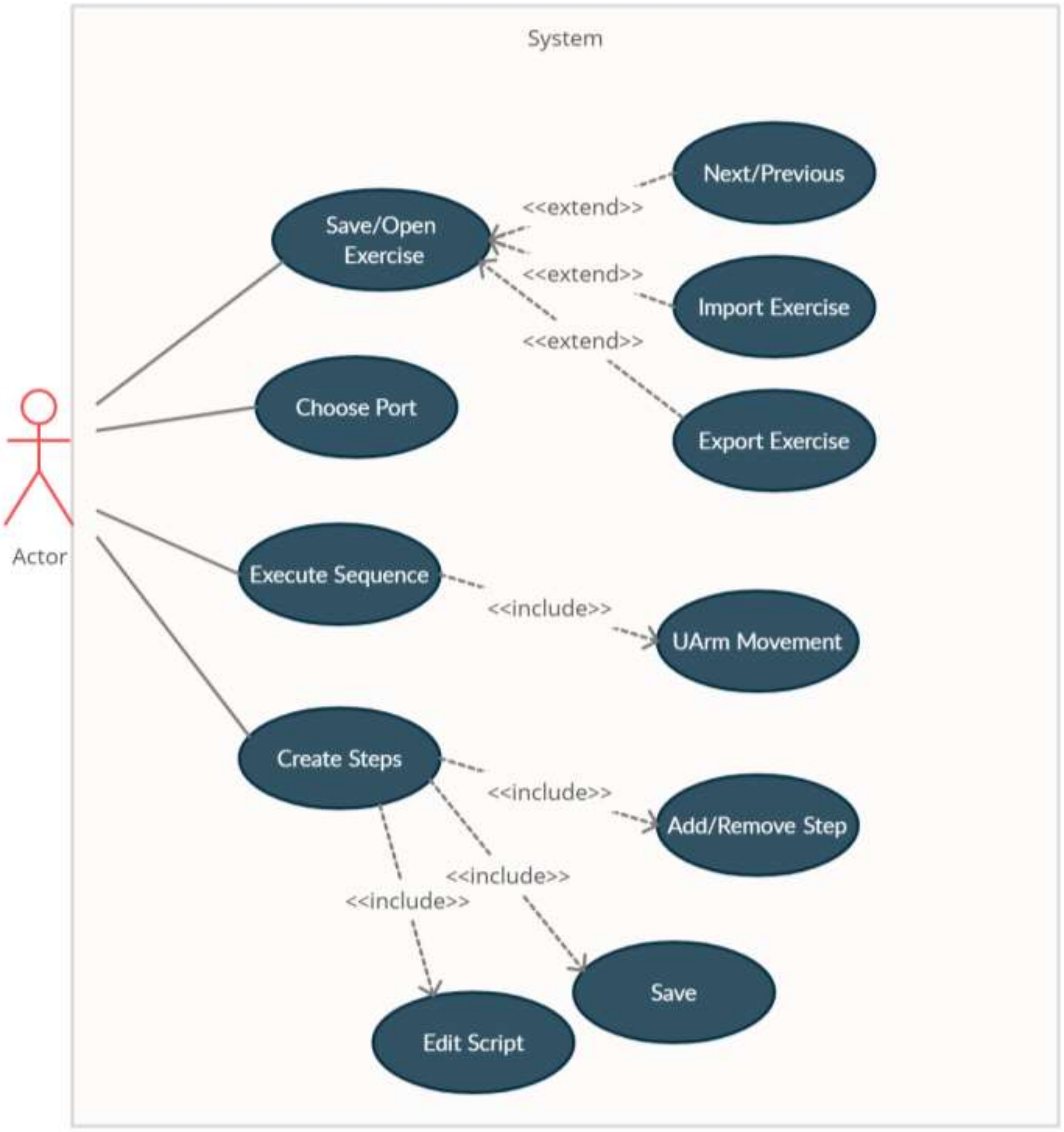
The following Use Case Diagrams are used to provide a perspective of the different interactions of the application with the user. Additionally, this was shown to the supervisor to better understand and improve on the application being developed.

Figure 24- Use Case Diagram #1



The following figure is the updated version of the Use Case Diagram after reviewing it with the supervisor. This updated version would simply various steps, making it easier to understand. This would also allow me to understand the interactions between the user and the methods in my application.

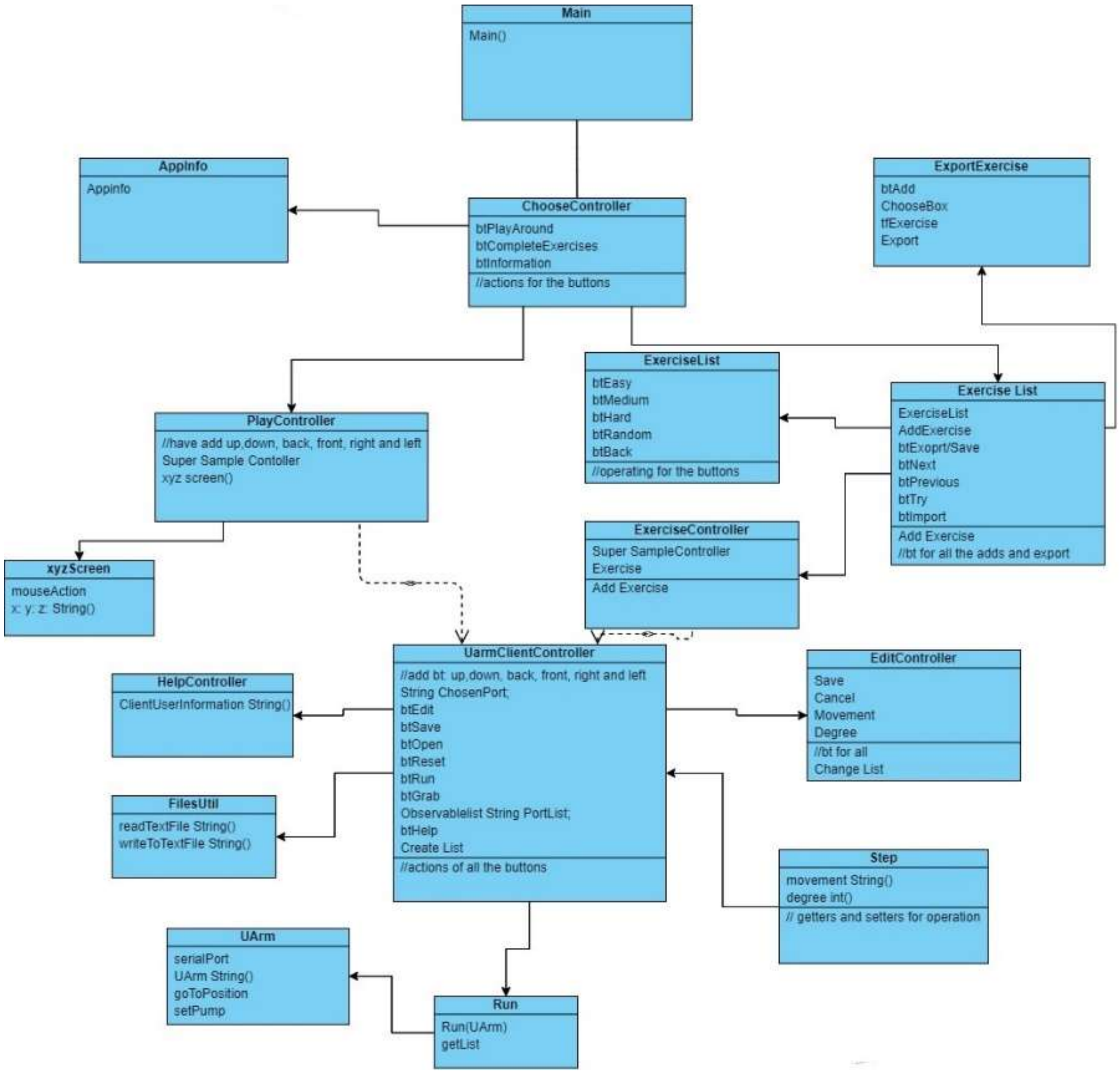
Figure 25- Use Case Diagram #2 (Updated)



Model of the Problem

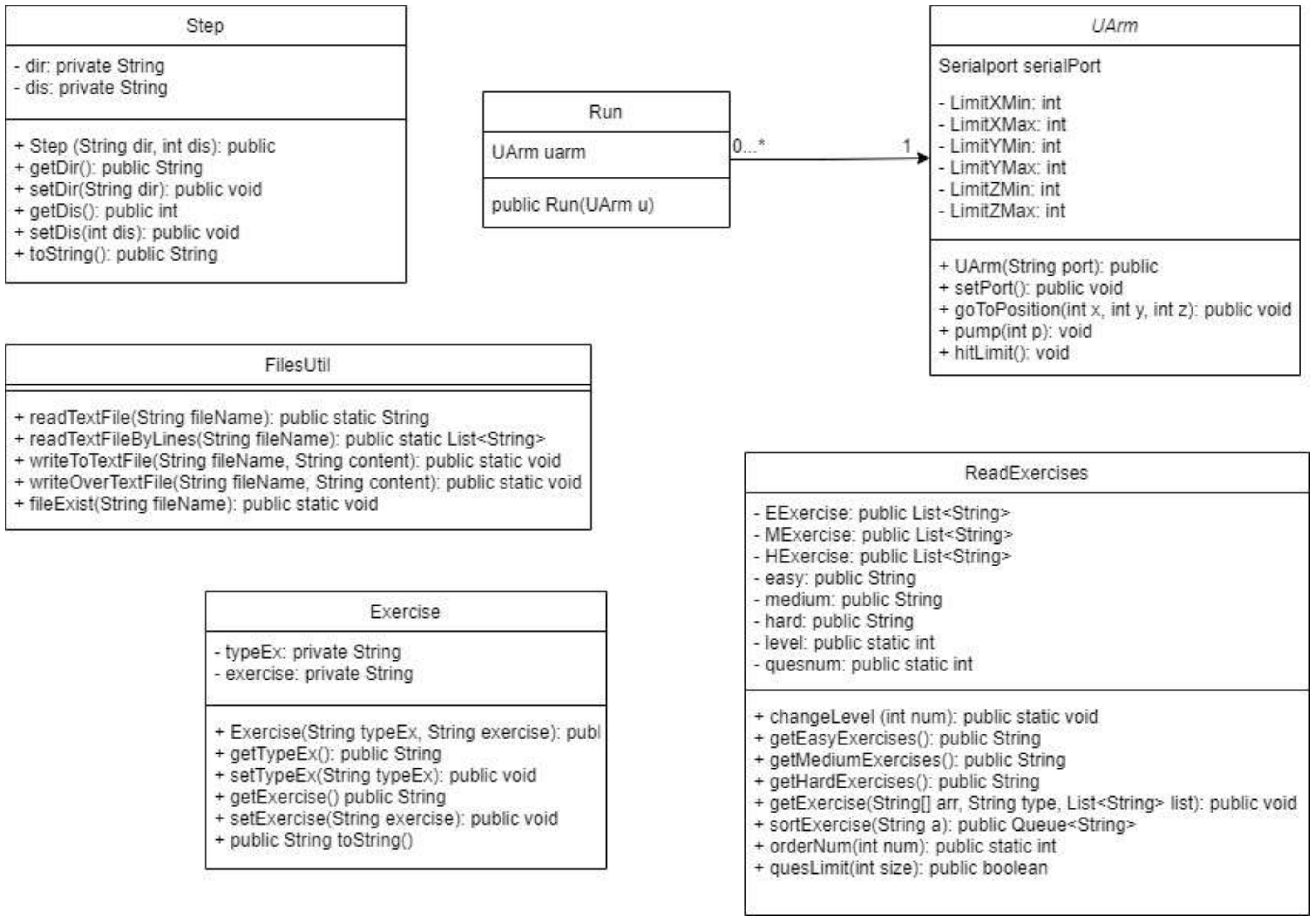
The following UML diagram is the overall model of the application that would help the client and I to better understand through a visual representation of the relationships between classes as well as the GUIs.

Figure 26- Overall View of Application



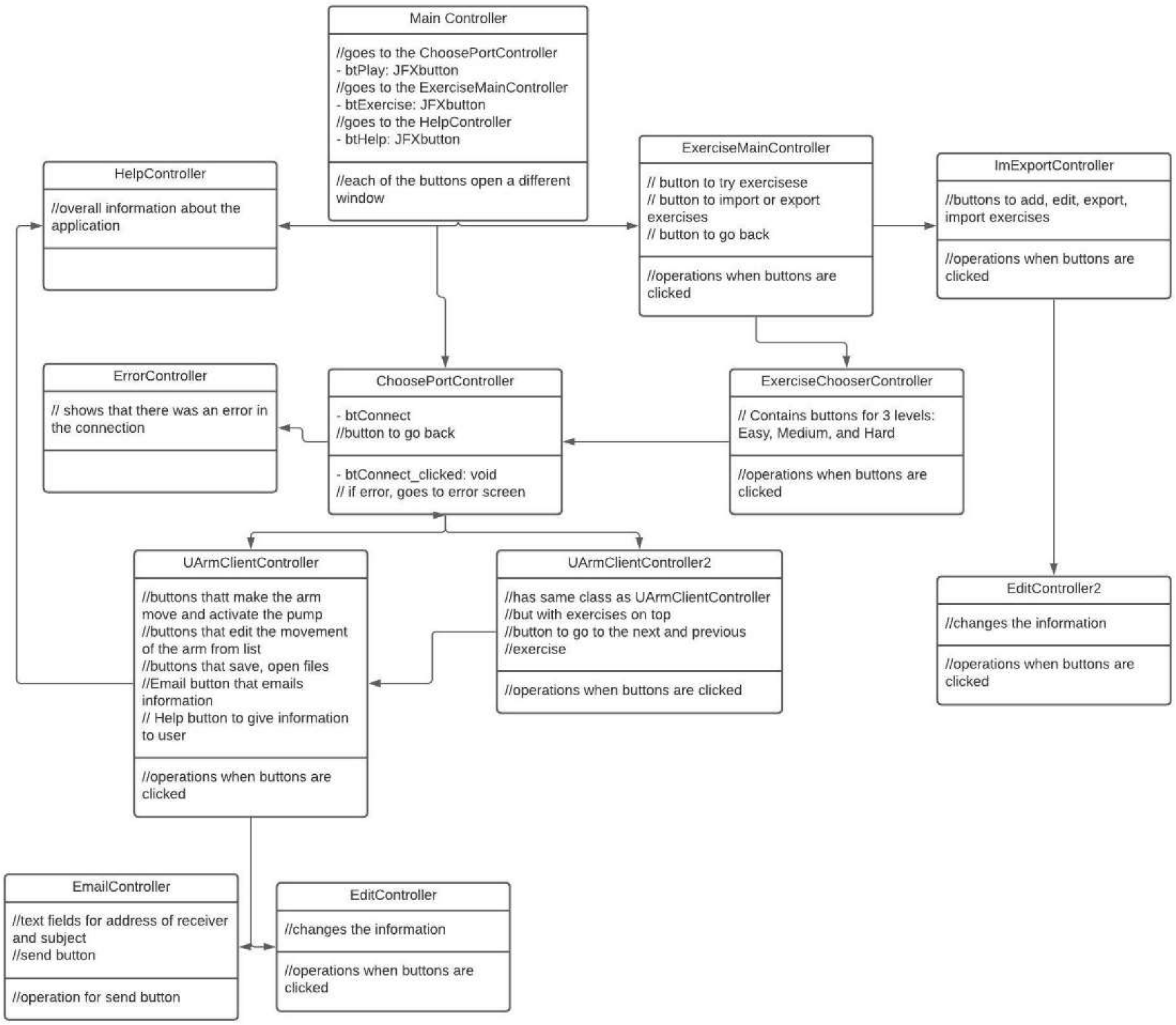
The following UML diagram presents classes being used in the application, which only refers to certain classes and not the controllers of the GUIs.

Figure 27- The Application Model (Classes)


Step
- dir: private String
- dis: private String
+ Step (String dir, int dis): public
+ getDir(): public String
+ setDir(String dir): public void
+ getDis(): public int
+ setDis(int dis): public void
+ toString(): public String
Run
UArm uarm
public Run(UArm u)
0...*
1
UArm
Serialport serialPort
- LimitXMin: int
- LimitXMax: int
- LimitYMin: int
- LimitYMax: int
- LimitZMin: int
- LimitZMax: int
+ UArm(String port): public
+ setPort(): public void
+ goToPosition(int x, int y, int z): public void
+ pump(int p): void
+ hitLimit(): void
FilesUtil
+ readTextFile(String fileName): public static String
+ readTextFileByLines(String fileName): public static List<String>
+ writeToTextFile(String fileName, String content): public static void
+ writeOverTextFile(String fileName, String content): public static void
+ fileExist(String fileName): public static void
Exercise
- typeEx: private String
- exercise: private String
+ Exercise(String typeEx, String exercise): publ
+ getTypeEx(): public String
+ setTypeEx(String typeEx): public void
+ getExercise() public String
+ setExercise(String exercise): public void
+ public String toString()
ReadExercises
- EExercise: public List<String>
- MExercise: public List<String>
- HExercise: public List<String>
- easy: public String
- medium: public String
- hard: public String
- level: public static int
- quesnum: public static int
+ changeLevel (int num): public static void
+ getEasyExercises(): public String
+ getMediumExercises(): public String
+ getHardExercises(): public String
+ getExercise(String[] arr, String type, List<String> list): public void
+ sortExercise(String a): public Queue<String>
+ orderNum(int num): public static int
+ quesLimit(int size): public boolean

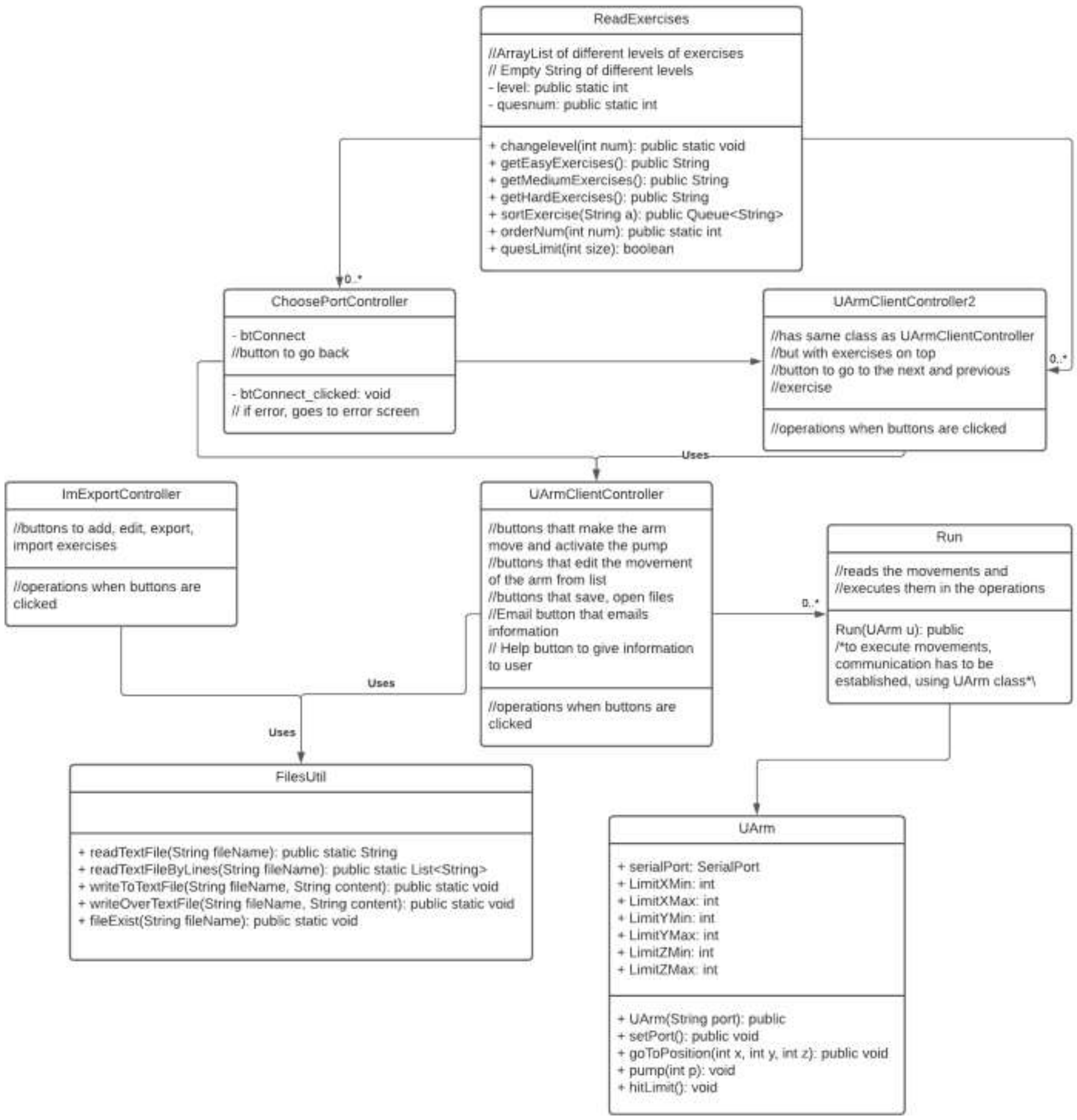
The following UML diagram shows the dependencies between the controllers, as it representing what is occurring when the windows are being switched or opened.

Figure 28- Dependencies between Controllers (managing GUIs)


Main Controller
//goes to the ChoosePortController
- btPlay: JFXbutton
//goes to the ExerciseMainController
- btExercise: JFXbutton
//goes to the HelpController
- btHelp: JFXbutton
//each of the buttons open a different window
HelpController
//overall information about the application
ExerciseMainController
// button to try exercisese
// button to import or export exercises
// button to go back
//operations when buttons are clicked
ImExportController
//buttons to add, edit, export, import exercises
//operations when buttons are clicked
ErrorController
// shows that there was an error in the connection
ChoosePortController
- btConnect
//button to go back
- btConnect_clicked: void
// if error, goes to error screen
ExerciseChooserController
// Contains buttons for 3 levels: Easy, Medium, and Hard
//operations when buttons are clicked
UArmClientController
//buttons thatt make the arm move and activate the pump
//buttons that edit the movement of the arm from list
//buttons that save, open files
//Email button that emails information
// Help button to give information to user
//operations when buttons are clicked
UArmClientController2
//has same class as UArmClientController
//but with exercises on top
//button to go to the next and previous
//exercise
//operations when buttons are clicked
EditController2
//changes the information
//operations when buttons are clicked
EmailController
//text fields for address of receiver and subject
//send button
//operation for send button
EditController
//changes the information
//operations when buttons are clicked

The following figure shows the relationships between the methods used and their data to improve the functionality and organization of the application.

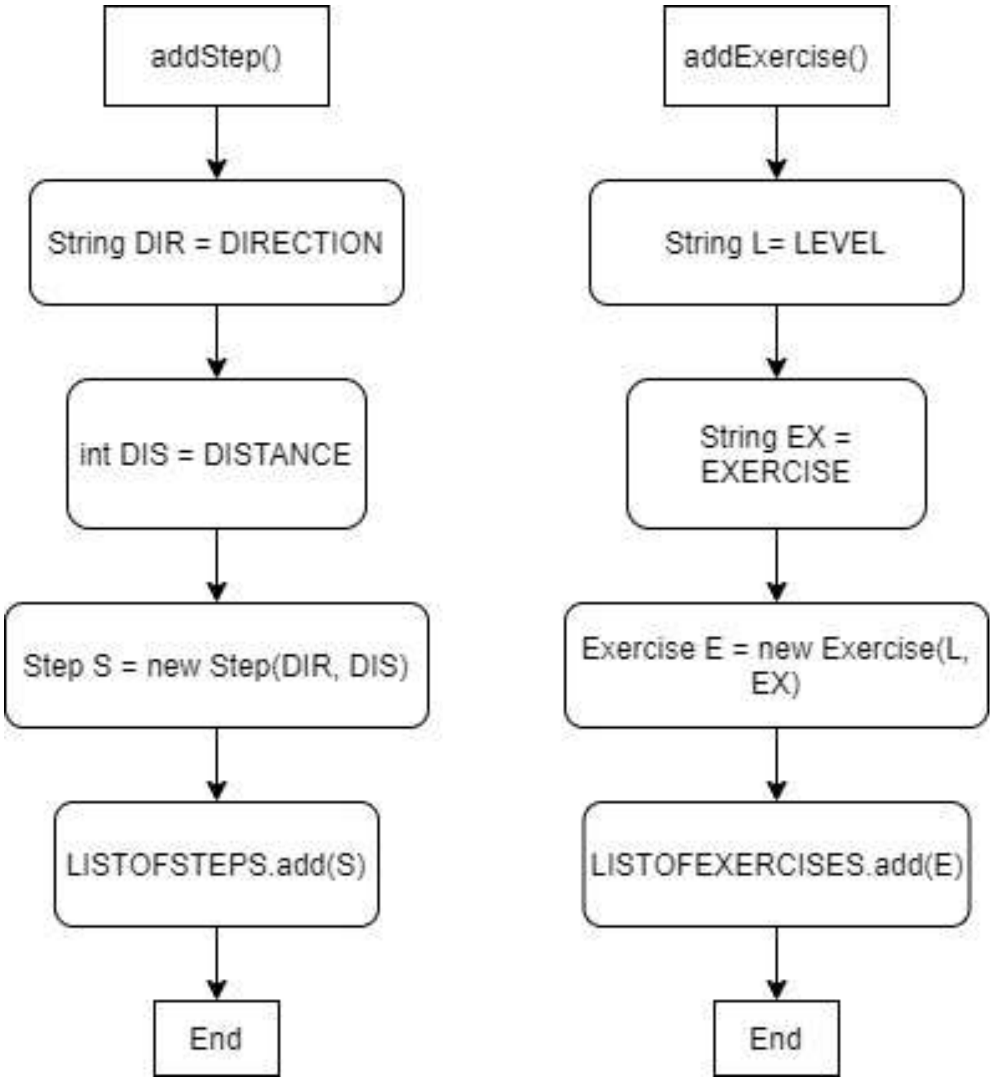
Figure 29- Relationships between Certain Classes and Controller


ReadExercises
//ArrayList of different levels of exercises
// Empty String of different levels
- level: public static int
- quesnum: public static int
+ changelevel(int num): public static void
+ getEasyExercises(): public String
+ getMediumExercises(): public String
+ getHardExercises(): public String
+ sortExercise(String a): public Queue<String>
+ orderNum(int num): public static int
+ quesLimit(int size): boolean
0..*
ChoosePortController
- btConnect
//button to go back
- btConnect_clicked: void
// if error, goes to error screen
UArmClientController2
//has same class as UArmClientController
//but with exercises on top
//button to go to the next and previous
//exercise
//operations when buttons are clicked
0..*
Uses
ImExportController
//buttons to add, edit, export, import exercises
//operations when buttons are clicked
UArmClientController
//buttons thatt make the arm move and activate the pump
//buttons that edit the movement of the arm from list
//buttons that save, open files
//Email button that emails information
// Help button to give information to user
//operations when buttons are clicked
Run
//reads the movements and
//executes them in the operations
Run(UArm u): public
/*to execute movements, communication has to be established, using UArm class*\
0..*
Uses
Uses
FilesUtil
+ readTextFile(String fileName): public static String
+ readTextFileByLines(String fileName): public static List<String>
+ writeToTextFile(String fileName, String content): public static void
+ writeOverTextFile(String fileName, String content): public static void
+ fileExist(String fileName): public static void
UArm
+ serialPort: SerialPort
+ LimitXMin: int
+ LimitXMax: int
+ LimitYMin: int
+ LimitYMax: int
+ LimitZMin: int
+ LimitZMax: int
+ UArm(String port): public
+ setPort(): public void
+ goToPosition(int x, int y, int z): public void
+ pump(int p): void
+ hitLimit(): void

Development of Algorithms

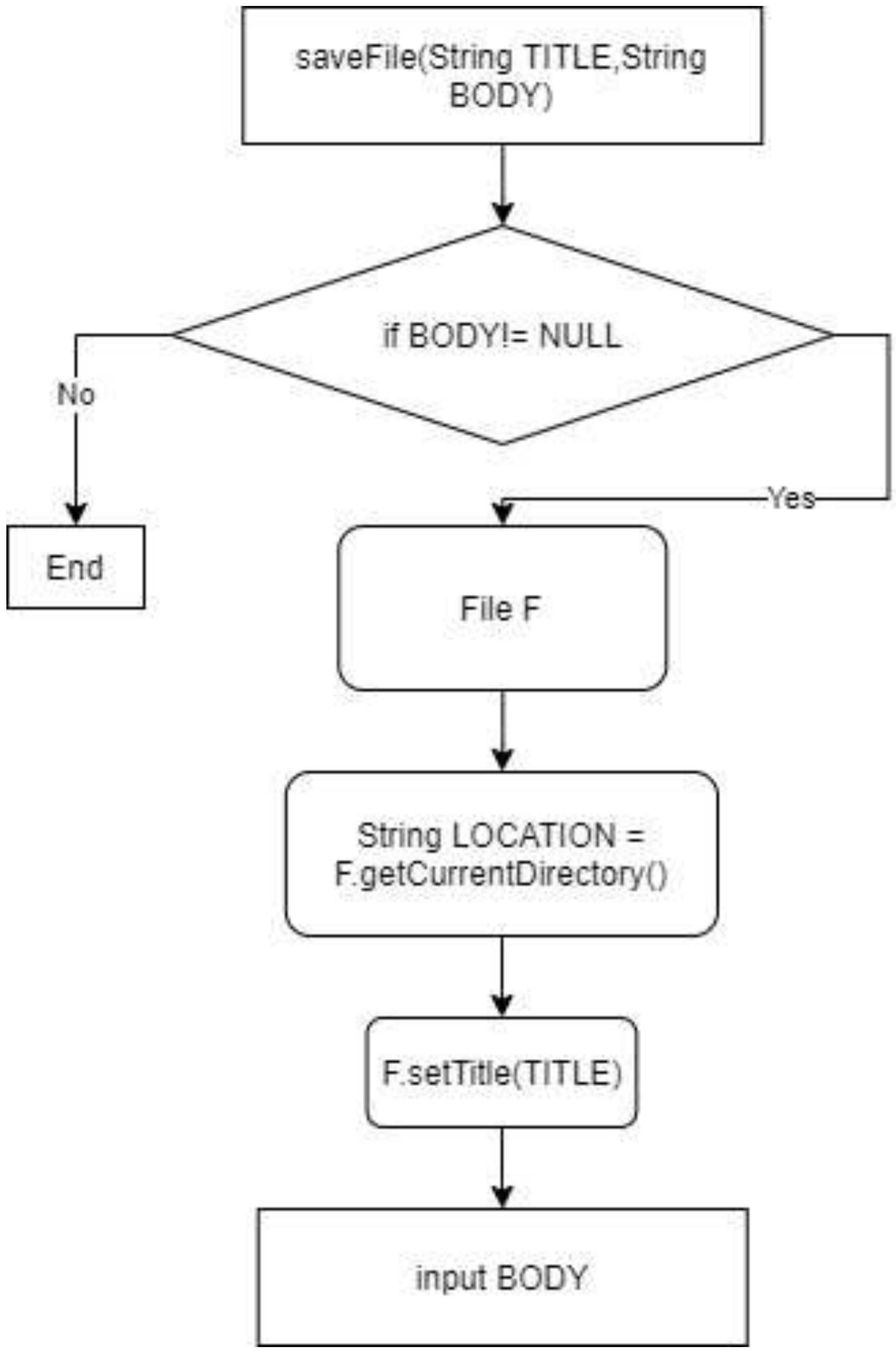
The following methods are to add steps (movement of the uArm) to the List of Steps and exercises (levels: easy, medium, hard) to be completed from the List of Exercises.

Figure 30- adding steps and exercises



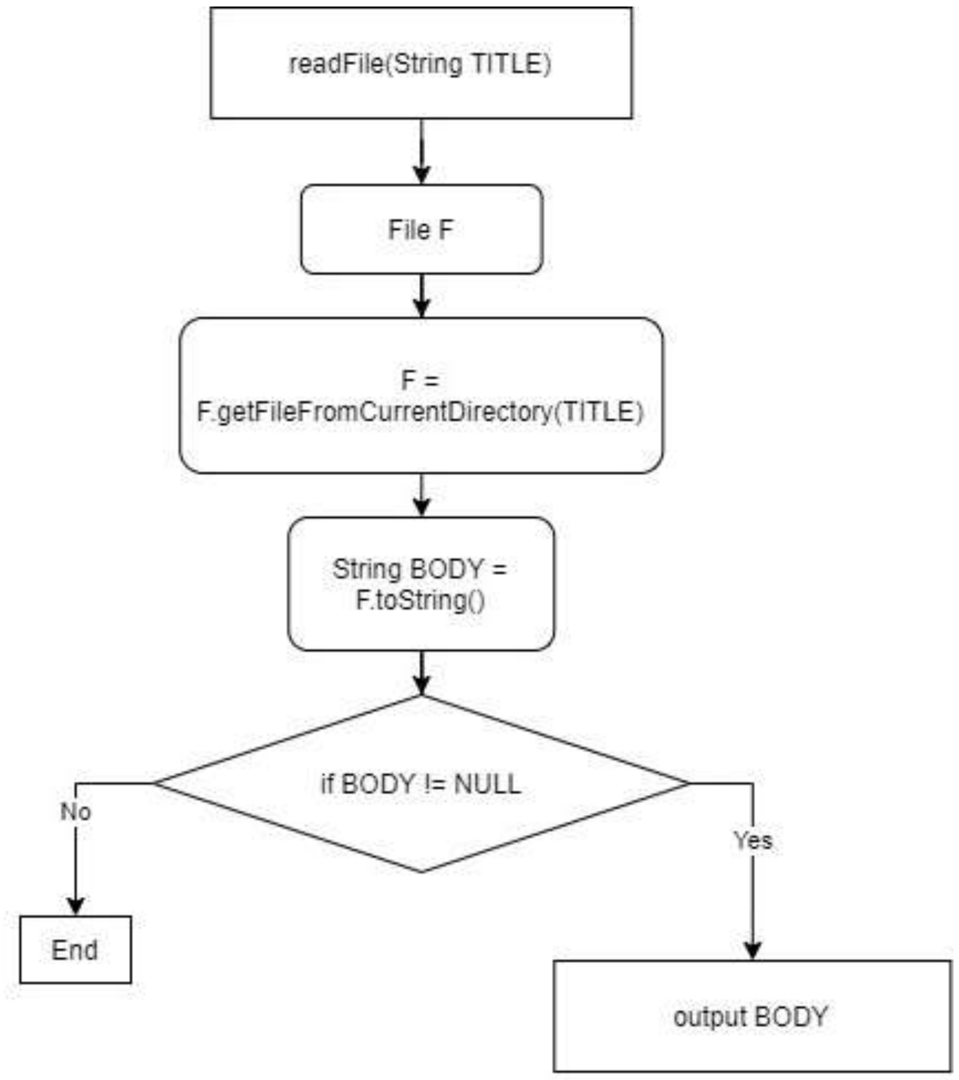
The following method will save a string to the directory as a text document (.txt file).

Figure 31- Saving Files


saveFile(String TITLE,String BODY)
if BODY!= NULL
No
Yes
End
File F
String LOCATION = F.getCurrentDirectory()
F.setTitle(TITLE)
input BODY

The following method will read a specific text document (.txt file) from the directory, so that the application can use it.

Figure 32- Reading Files

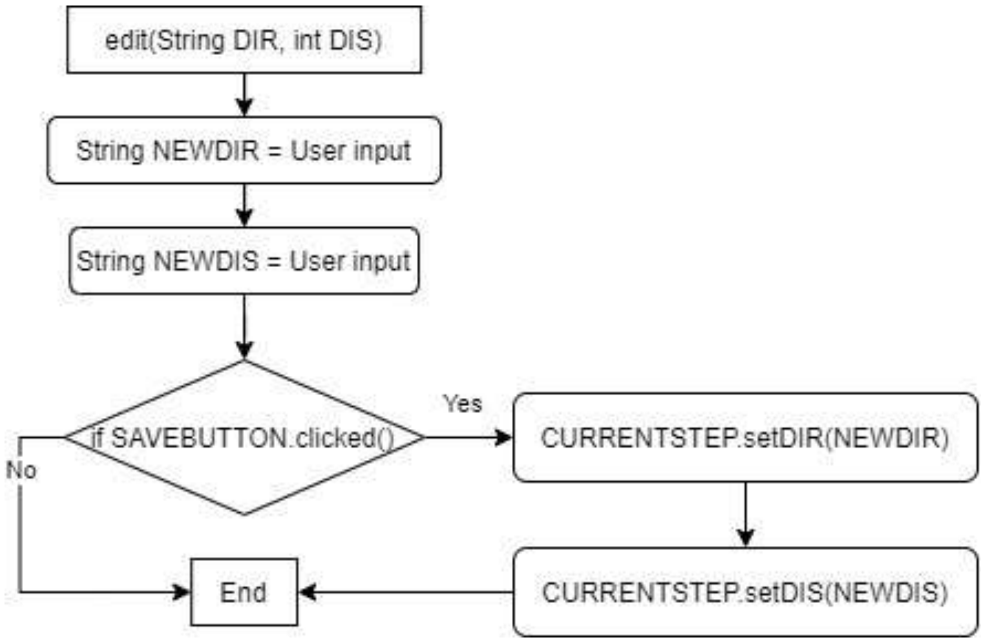

readFile(String TITLE)
File F
F = F.getFileFromCurrentDirectory(TITLE)
String BODY = F.toString()
if BODY != NULL
No
End
Yes
output BODY

The saving and reading files methods are used as helper methods, since the user is able to change their own data that they have inputted as well as being able to add, edit or delete from the list, as it's being saved into a file in the directory and read by the application making it more user-friendly and scalable.

Method	Input	Output
saveFile(String TITLE, String BODY)	String TITLE = "ListOfSteps" String BODY = "Right:10, Left:20"	ListOfSteps.txt with the body of "Right:10, Left:20"
readFile(String TITLE)	String TITLE = "ListOfSteps"	"Right:10, Left:20"

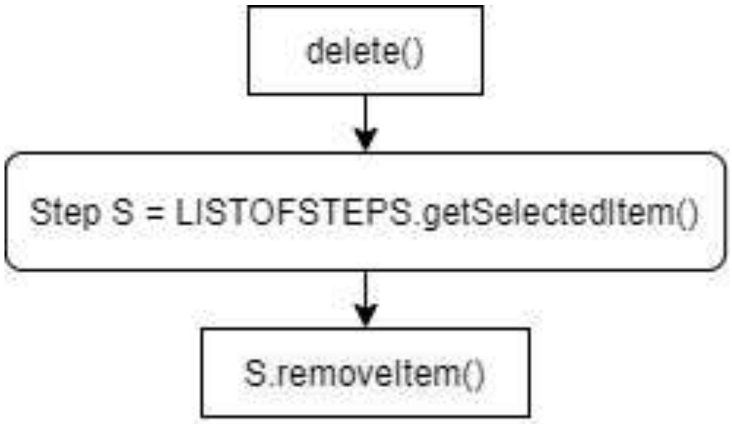
The following method would edit a step/exercise.

Figure 33- Edit Step or Exercise


edit(String DIR, int DIS)
String NEWDIR = User input
String NEWDIS = User input
if SAVEBUTTON.clicked()
Yes
No
CURRENTSTEP.setDIR(NEWDIR)
CURRENTSTEP.setDIS(NEWDIS)
End

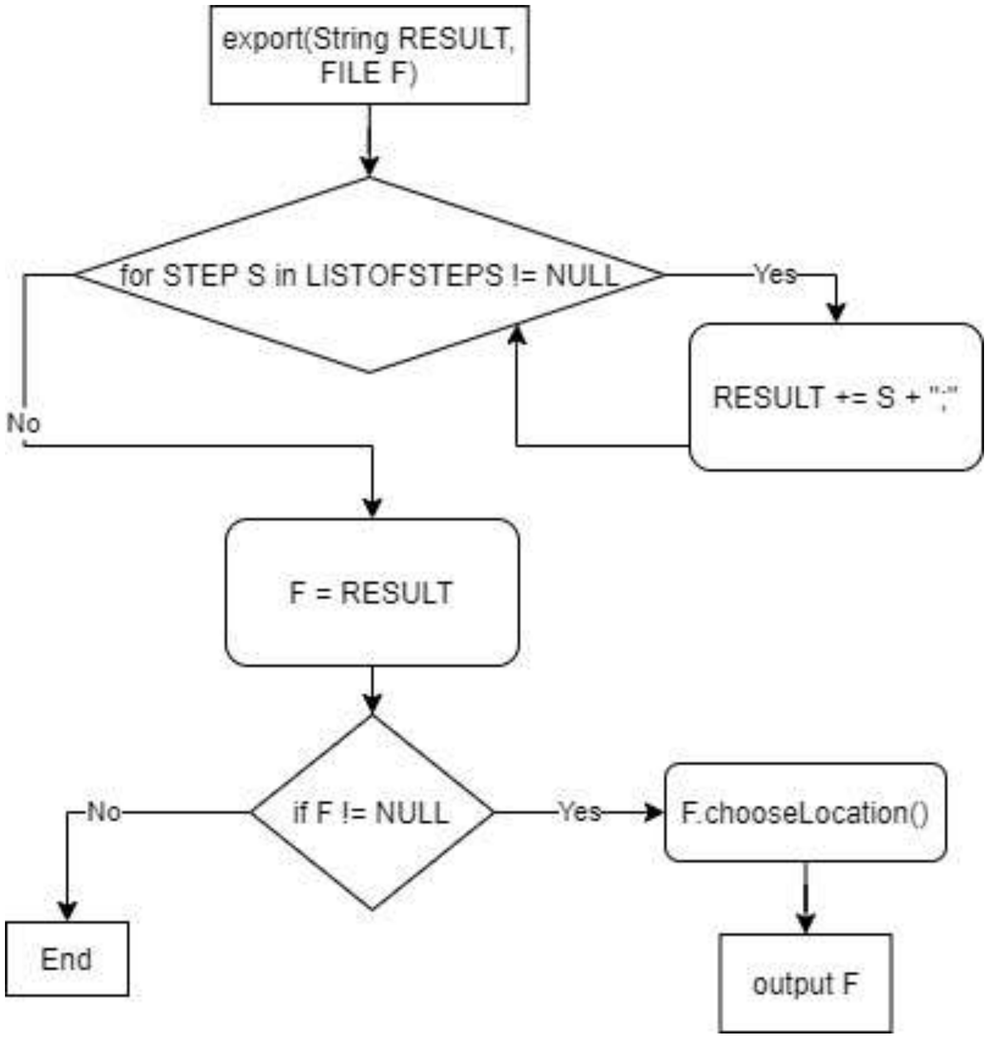
Additionally, the following method deletes the step/exercise from the list.

Figure 34- Delete Step or Exercise



The following method is to export the list of steps/exercises, as a text document (.txt) with a single String to any destination the user chooses.

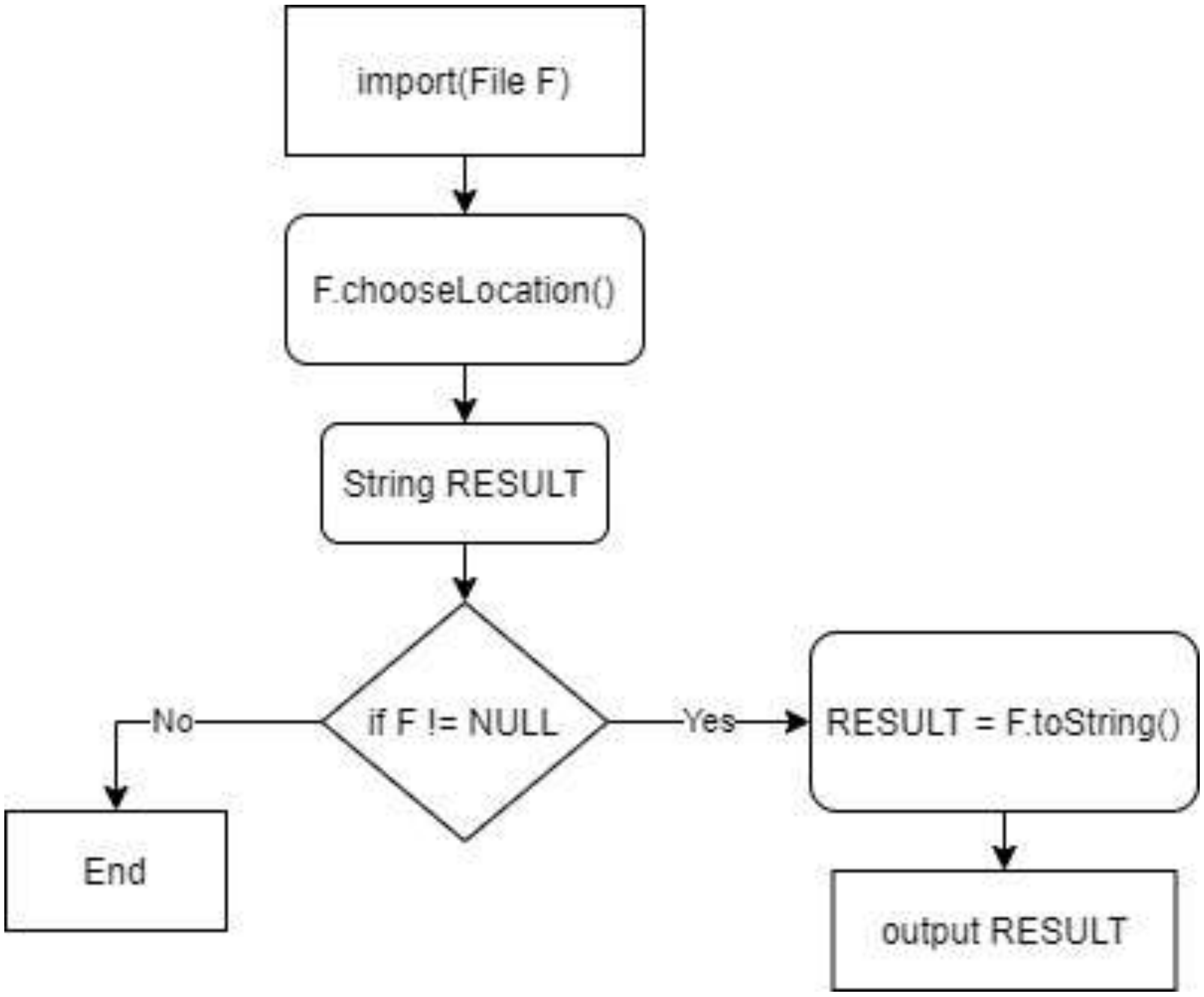
Figure 35- Exporting File



The .chooseLocation() would be a method that allows the user chose a location in their computers. This is similar to the next method.

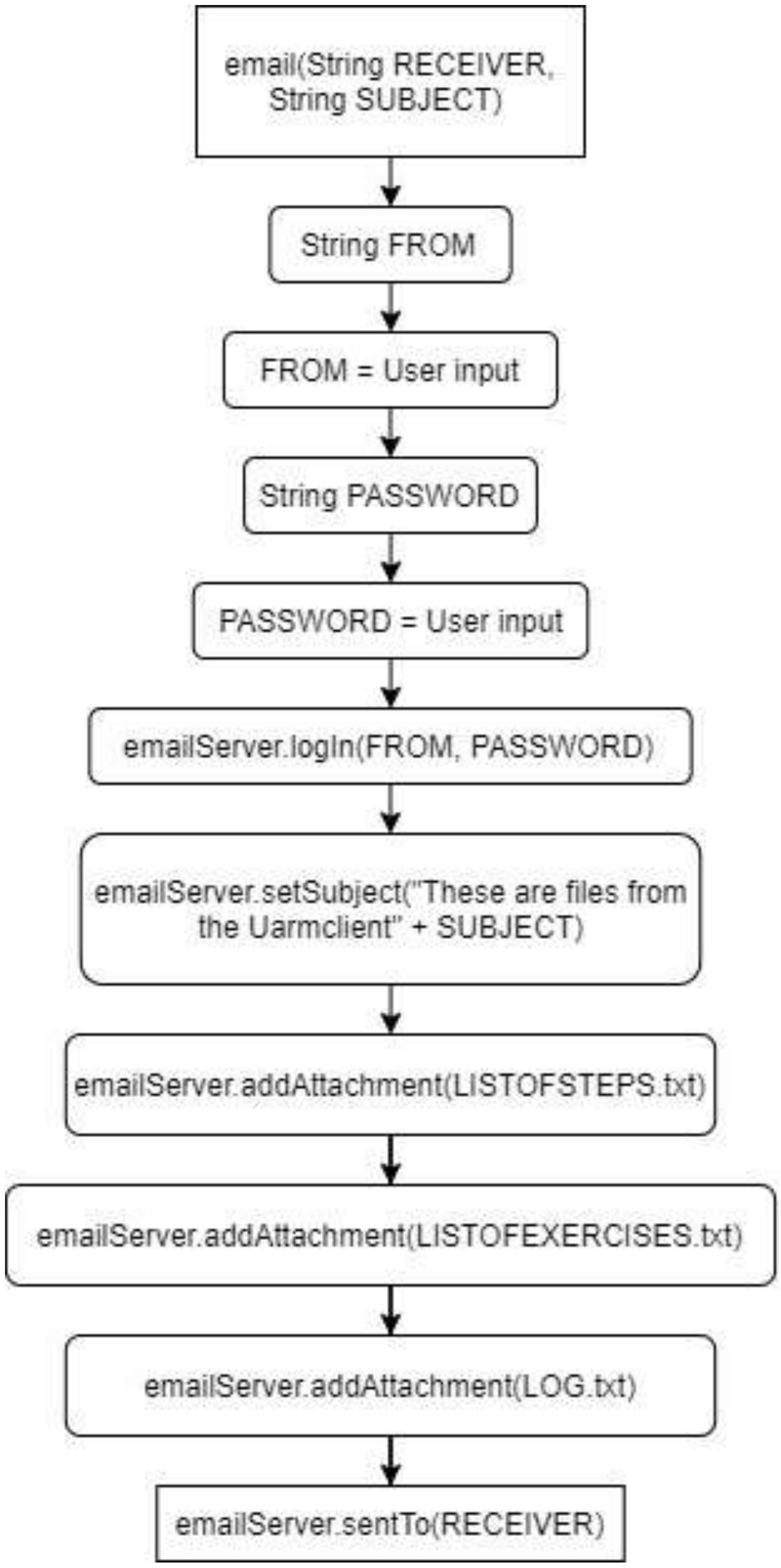
The following method is to import a list of steps/exercises from a text document (.txt) into the application's current directory, read by the readFile() method.

Figure 36- Importing File



The following method will send an email of all the information in the application.

Figure 37- Email information in application

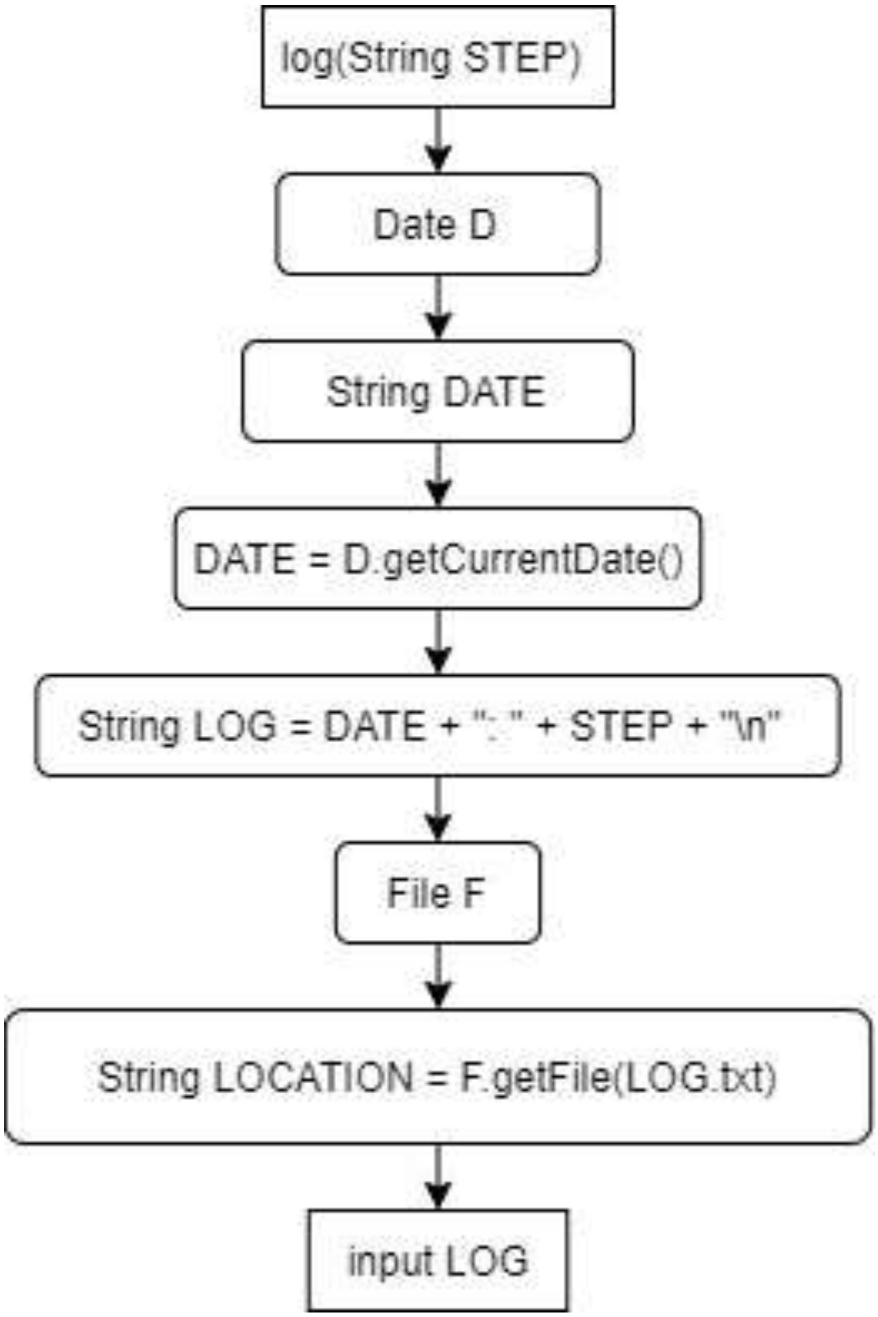


email(String RECEIVER, String SUBJECT)
String FROM
FROM = User input
String PASSWORD
PASSWORD = User input
emailServer.logIn(FROM, PASSWORD)
emailServer.setSubject("These are files from the Uarmclient" + SUBJECT)
emailServer.addAttachment(LISTOFSTEPS.txt)
emailServer.addAttachment(LISTOFEXERCISES.txt)
emailServer.addAttachment(LOG.txt)
emailServer.sentTo(RECEIVER)

The email server would be classes (a package) that would be implemented in the application

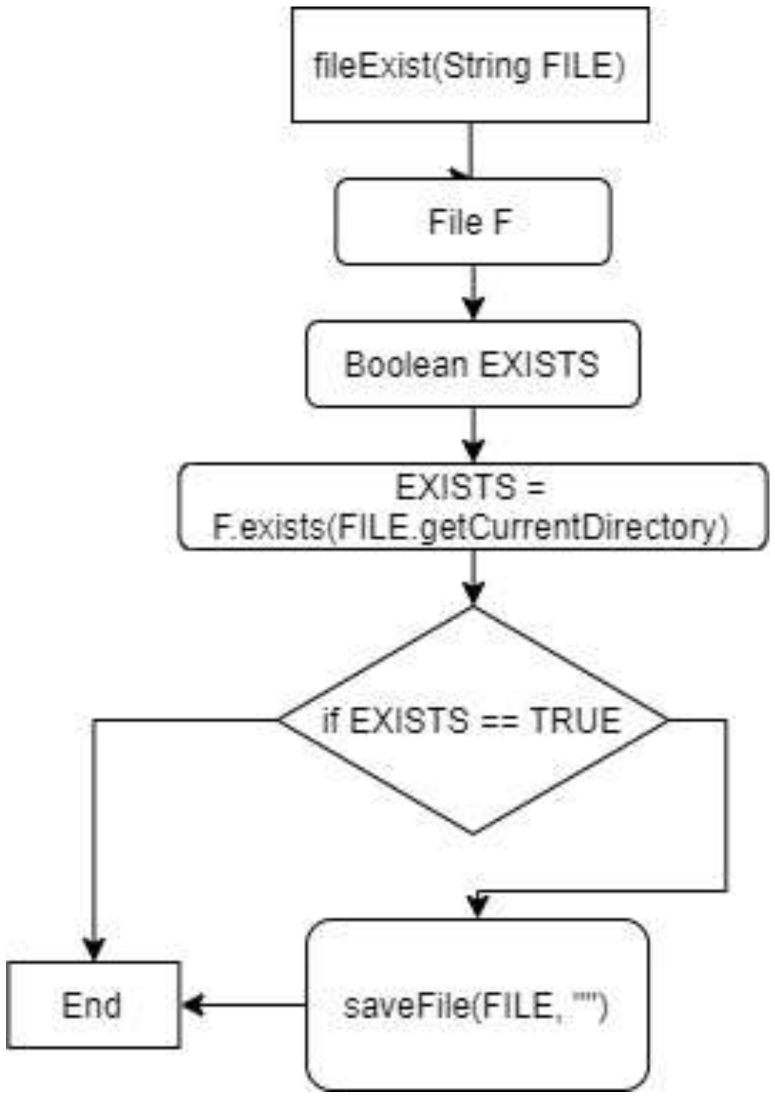
The following method creates a log of what the steps executed by the application.

Figure 38- Logging steps


log(String STEP)
Date D
String DATE
DATE = D.getCurrentDate()
String LOG = DATE + ": " + STEP + "\n"
File F
String LOCATION = F.getFile(LOG.txt)
input LOG

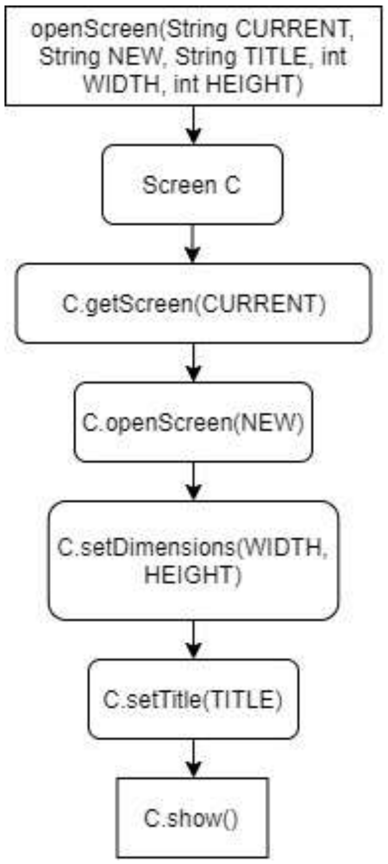
The following method would check if the file exists (.txt file) in the current directory, if not, then it restores it. Using a relative path instead of the absolute path to improve the functionality of the application.

Figure 39- Checking if file exists



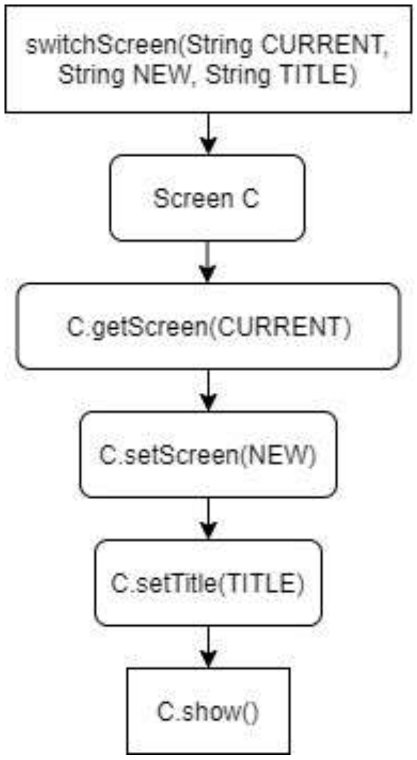
The following method would open a new window.

Figure 40- open Window


openScreen(String CURRENT, String NEW, String TITLE, int WIDTH, int HEIGHT)
Screen C
C.getScreen(CURRENT)
C.openScreen(NEW)
C.setDimensions(WIDTH, HEIGHT)
C.setTitle(TITLE)
C.show()

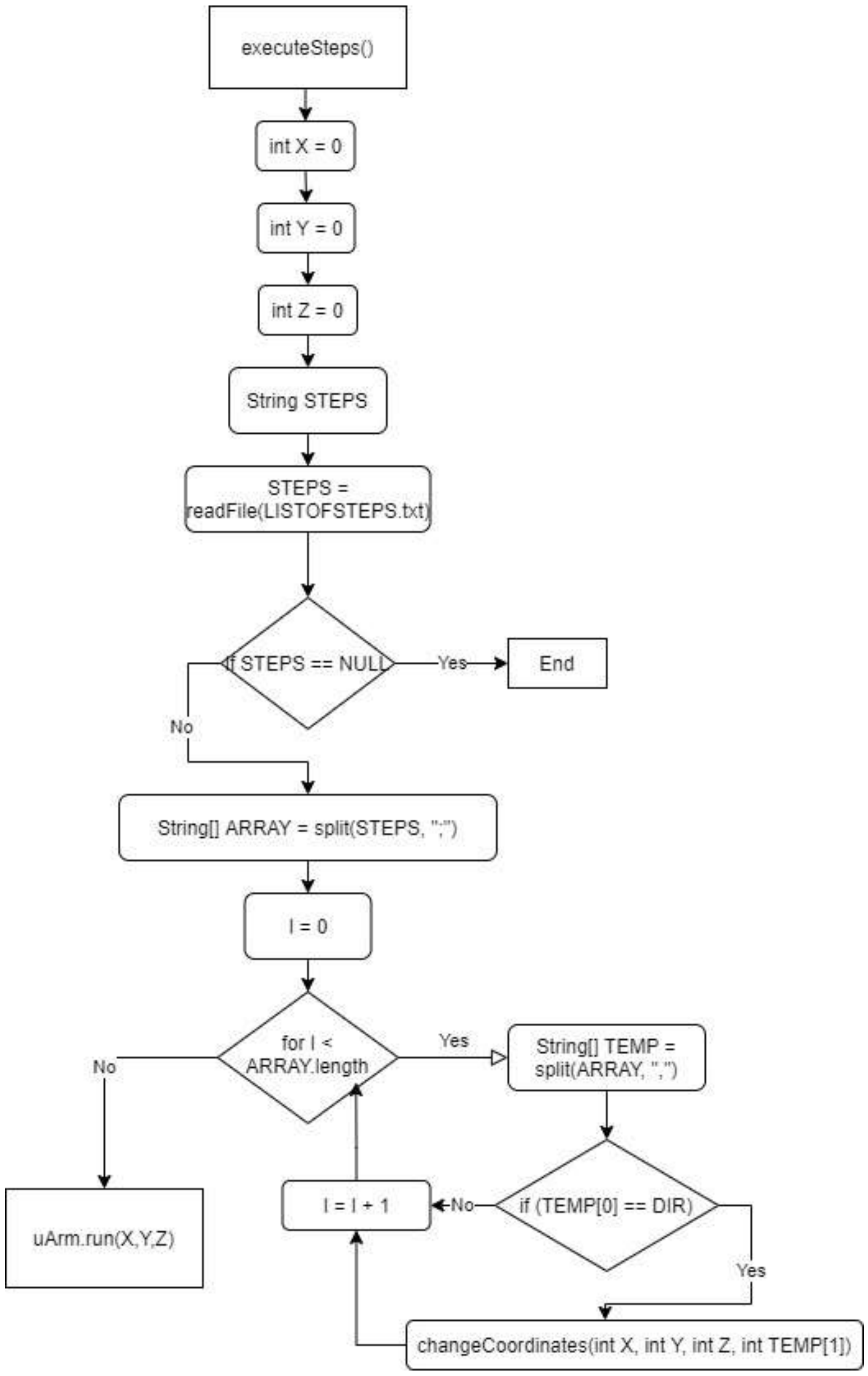
The following method would switch windows.

Figure 41- Switch Window



The following method will execute all the steps that are being read from the application and would communicate to the uArm, to make it move.

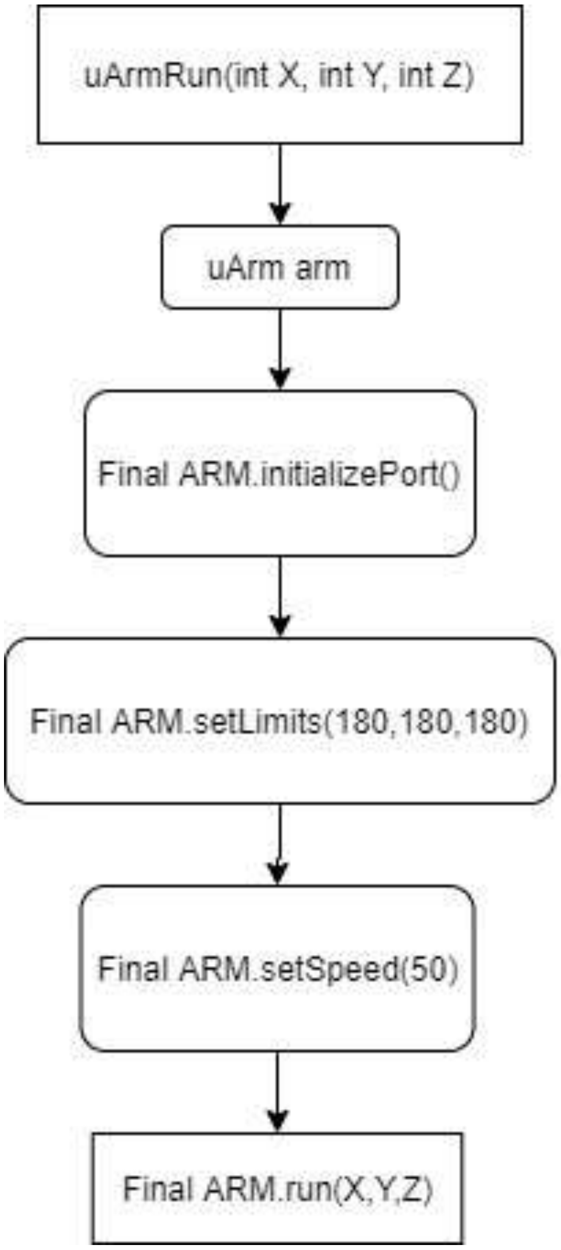
Figure 42- Executing Steps



The function changeCoordinates(int X, int Y, int Z, int TEMP[1]) would change the values of X, Y, or Z by adding or subtracting the integer, TEMP[1].

Additionally, the uArmRun(X,Y,Z), the following method that would communicate and move the arm in terms of the XYZ axis.

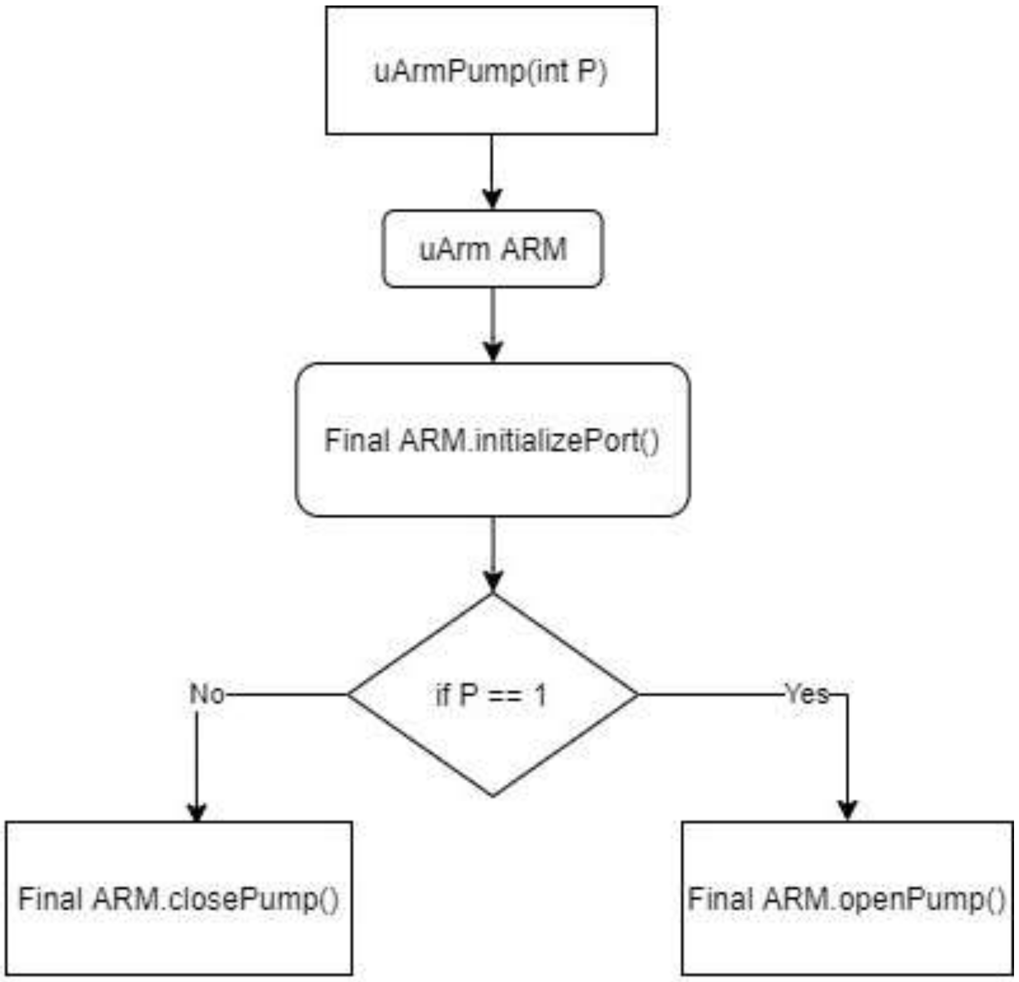
Figure 43- Moving uArm


uArmRun(int X, int Y, int Z)
uArm arm
Final ARM.initializePort()
Final ARM.setLimits(180,180,180)
Final ARM.setSpeed(50)
Final ARM.run(X,Y,Z)

Even though this seems simple, it’s the abstract version of what is going to be communicated with the robotic arm.

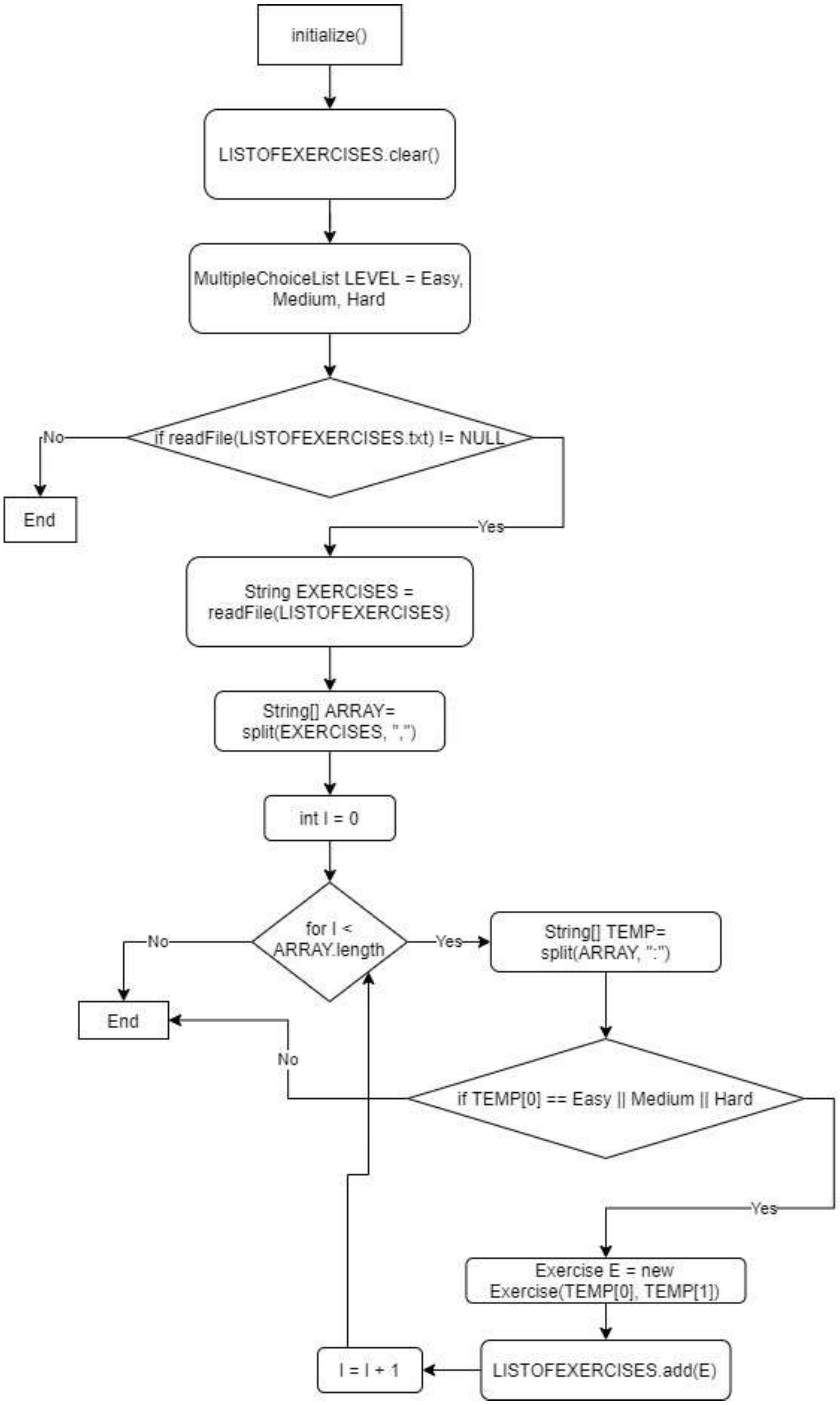
The following method would activate/disactivate a pump.

Figure 44- Activating or Disactivating the pump



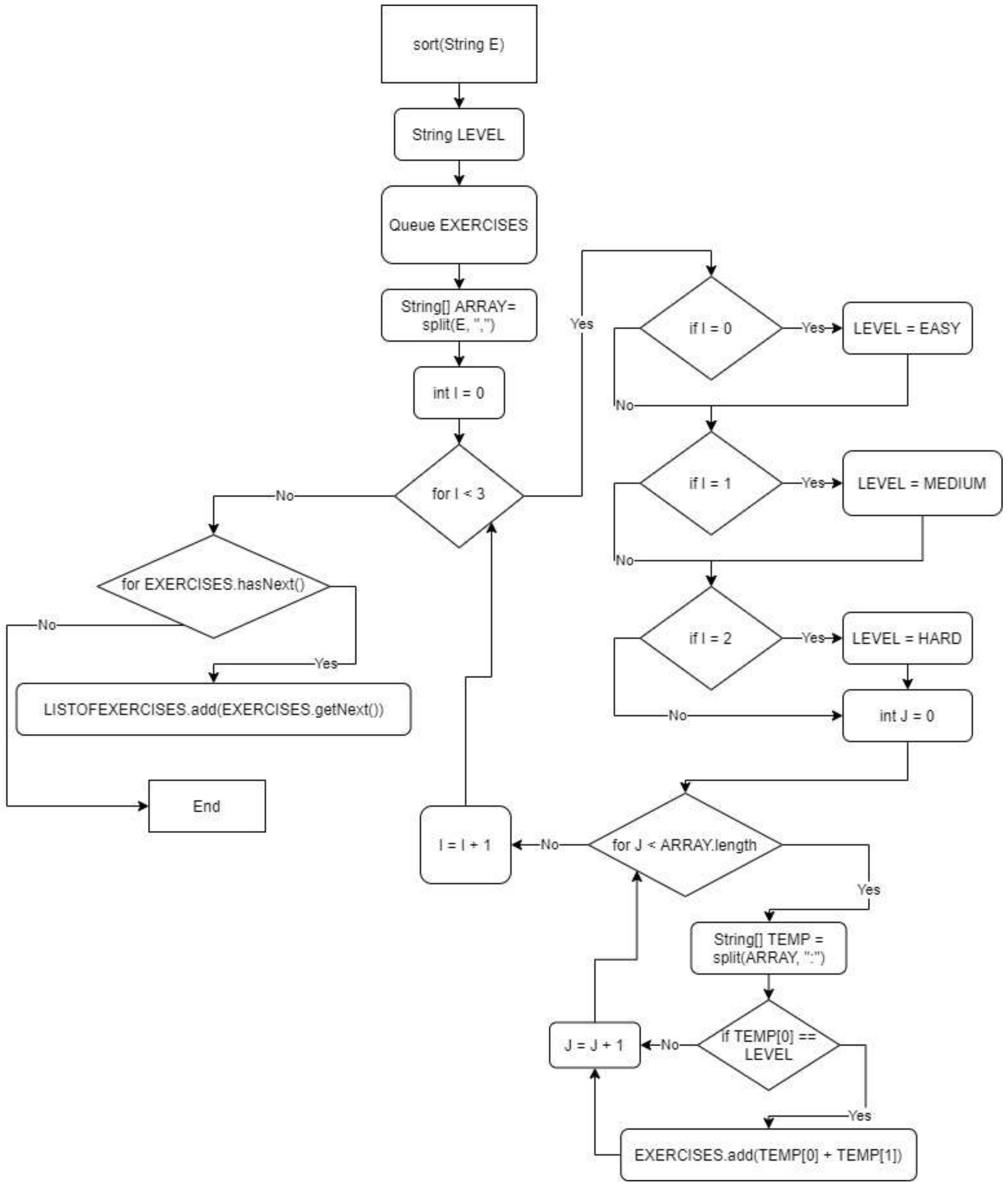
The following method will initialize the Import and Export Exercise Controller, for the list of exercises.

Figure 45- Initializing the Import and Export Controller


initialize()
LISTOFEXERCISES.clear()
MultipleChoiceList LEVEL = Easy, Medium, Hard
if readFile(LISTOFEXERCISES.txt) != NULL
No
End
Yes
String EXERCISES = readFile(LISTOFEXERCISES)
String[] ARRAY= split(EXERCISES, ",")
int I = 0
for I < ARRAY.length
No
End
Yes
String[] TEMP= split(ARRAY, ":")
No
if TEMP[0] == Easy || Medium || Hard
Yes
Exercise E = new Exercise(TEMP[0], TEMP[1])
I = I + 1
LISTOFEXERCISES.add(E)

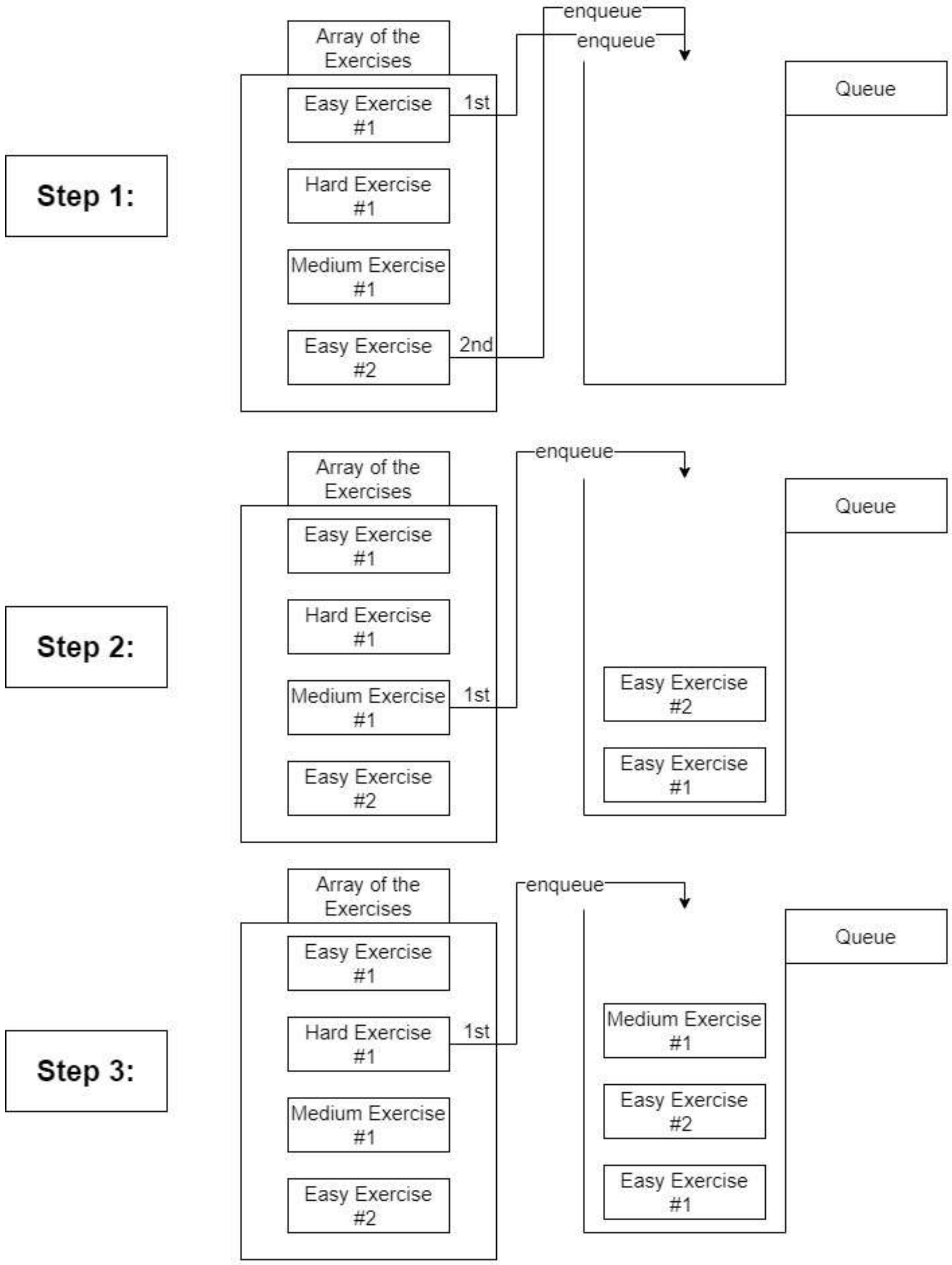
The following method would sort all the exercises based on their levels in the Import and Export Exercise Controller when new exercises are added to the list, so that it's always sorted.

Figure 46- Sorting list of Exercises

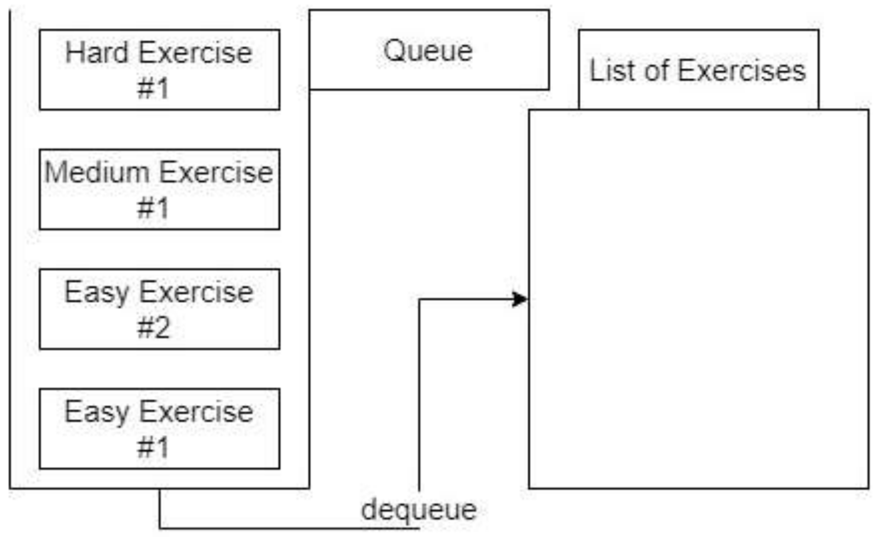


So, it should be something like this after splitting the string into an Array and enqueuing it into a Queue:

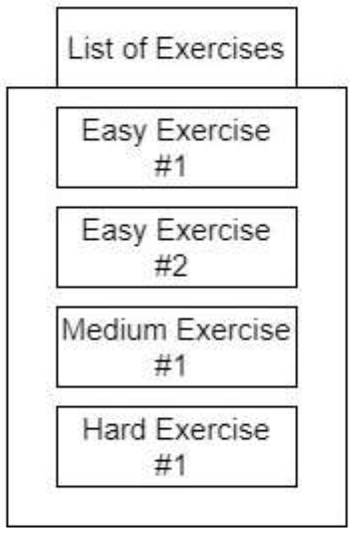
Figure 47- Illustration of Sorting Exercises



Step 4:

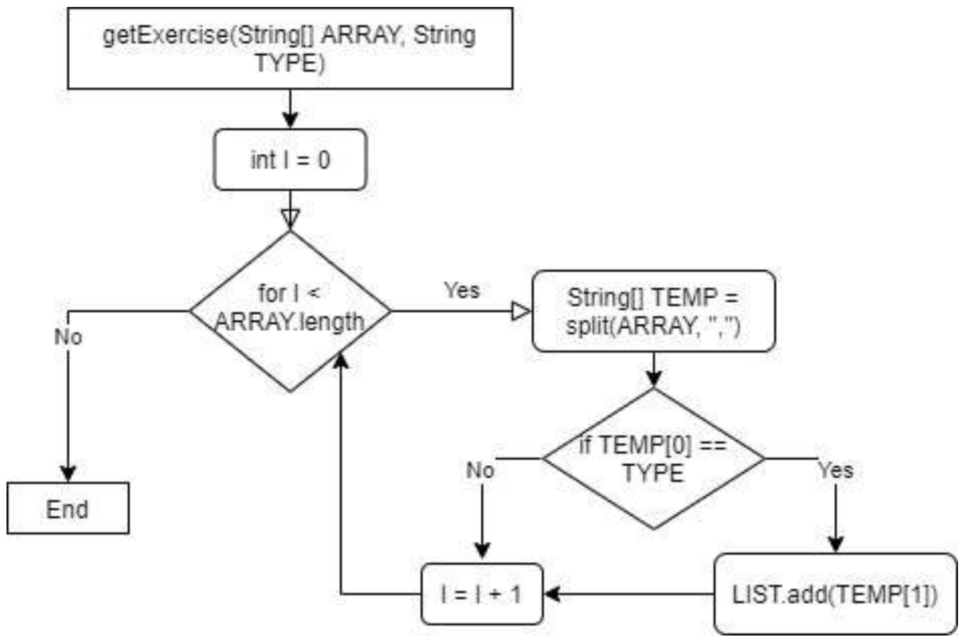

Hard Exercise #1
Medium Exercise #1
Easy Exercise #2
Easy Exercise #1
Queue
List of Exercises
dequeue

Final:


List of Exercises
Easy Exercise #1
Easy Exercise #2
Medium Exercise #1
Hard Exercise #1

The following method is to generate Lists with specific exercises.

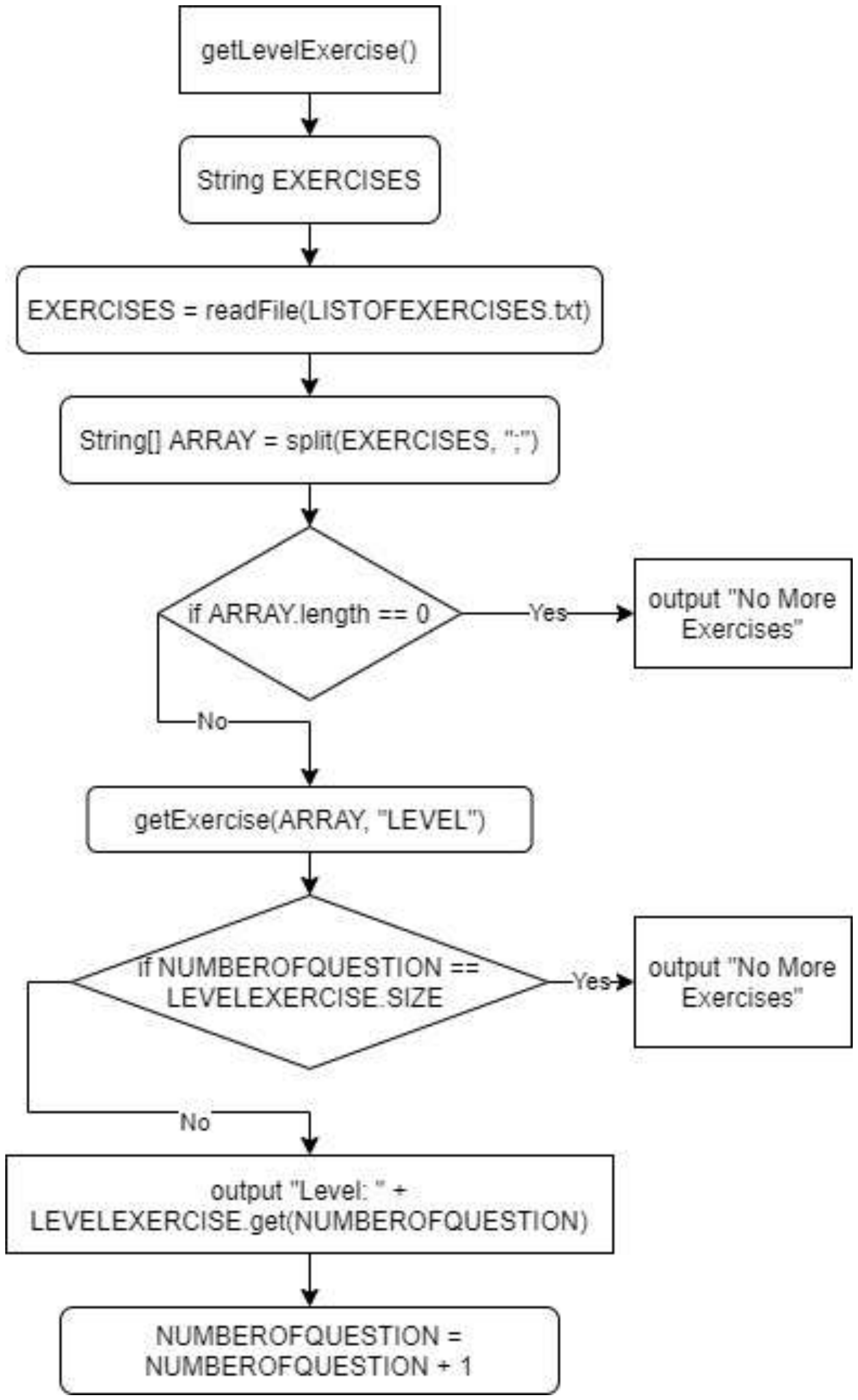
Figure 48- Get Exercise



The LIST would be the different level of lists (easy, medium, hard), referring to LEVELEXERCISE, for instance, EASYEXERCISE.

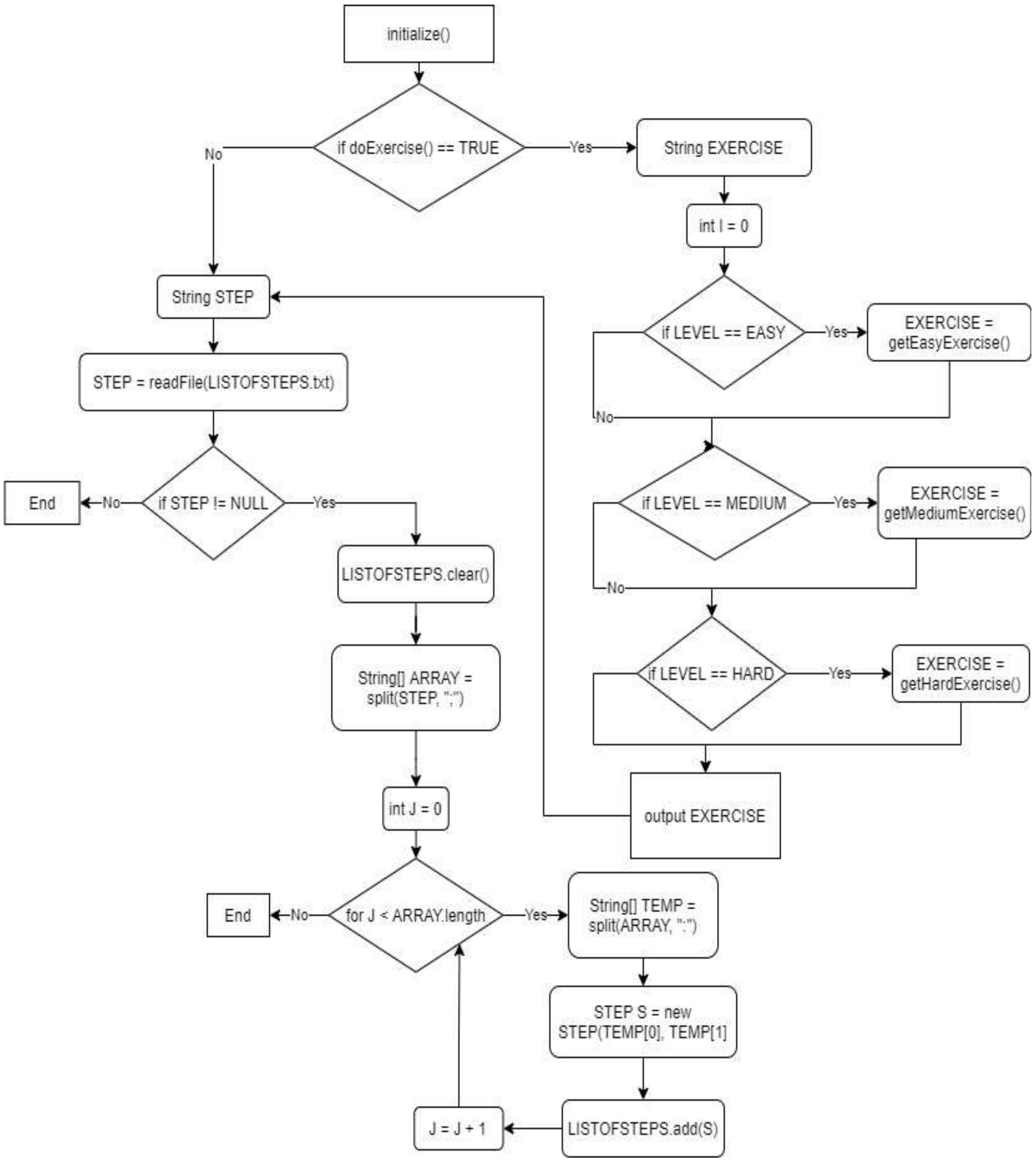
The following method will get the level of exercises and previous method in order to return an exercise at that level.

Figure 49- Get the Level of Exercise


getLevelExercise()
String EXERCISES
EXERCISES = readFile(LISTOFEXERCISES.txt)
String[] ARRAY = split(EXERCISES, ";")
if ARRAY.length == 0
Yes
output "No More Exercises"
No
getExercise(ARRAY, "LEVEL")
if NUMBEROFQUESTION == LEVELEXERCISE.SIZE
Yes
output "No More Exercises"
No
output "Level: " + LEVELEXERCISE.get(NUMBEROFQUESTION)
NUMBEROFQUESTION = NUMBEROFQUESTION + 1

The following method initializes each of the uArm Client Controllers (window where the user can control the arm) based on if there are doing exercises or just practicing, based on the level chosen.

Figure 50- initializing uArm Client



Test Plan

Actions to be tested	Test Method
Test if the application runs correctly	Export the application as a runnable jar file and check if all the functions in the application works properly.
See if the application is aware that the arm is connected to the computer	See if the serial ports connected appear in the application.
See if a connection can be established between the user and the arm	Connect the arm with the computer and see if the arm moves.
An error appears if no arm is connected	Try to connect to the arm without selecting the port that the arm is connected to, making an error appear.
The application can accurately send information to the arm.	Add steps and execute them (clicking run or reset) and see if the arm moves to the desired location.
The robotic arm is able to function properly	By clicking the run button to make the arm move, and click the grab button, to make sure the pump is working.
The user is able to understand how the application works	Reading over the information section to make sure it's understandable and is clear, could be done with an end user.
The application correctly adds the steps/exercises to the list	Create multiple steps/exercises with different characteristics and see if they appear on the list.
Save the steps/exercises as a text document to the current directory	Close the application and open it again to see if the list of steps is the same as previously run.
Deletes and Clears the list of steps/exercises	By clicking the delete or clear button, a step/exercise or all the step/exercise disappears from the list.
Editing the list of steps/exercises	By clicking the edit button when selecting a step/exercise, a window that allows the user to edit the step/exercise.
Saving the information in the application in a different location	By clicking the Save As/Exporting button, a window is opened that allows the user to save a text document to the location the user chooses.
Opens and reads the saved information saved on the computer	By clicking the Open/Importing button, a window is opened that allows the user to open a text document. After opening, all the elements in the text document in the list.

The List of Exercises are automatically sorted by level (easy, medium to hard)	By adding/editing an exercise, they go to their corresponding level in the list
Correct Exercises of different levels are shown corresponding to the level chosen	Exercises appear at the level the user chose.
The correct amount of exercises are available to the user	As the user goes through the exercises, by clicking next, they will eventually not be able to continue clicking as it outputs: “No Exercises Available”.
Application can work properly without preexisting files in the current directory	Deleting the text document files in the current directory of the application and then running the application, checking if the application is working properly and if the text documents are restored.
Emailing the information of the application to someone else	Clicking the email button and inputting the receiver and the subject. Check if the receiver received the list of steps, exercises, and log of the application.