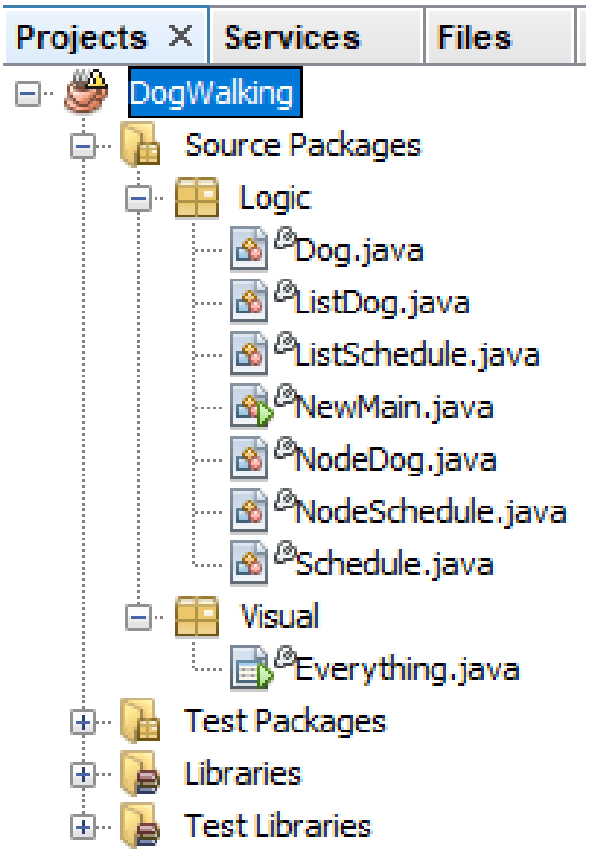
Criterion C:

Organization of Classes:



BackEnd (Logic)

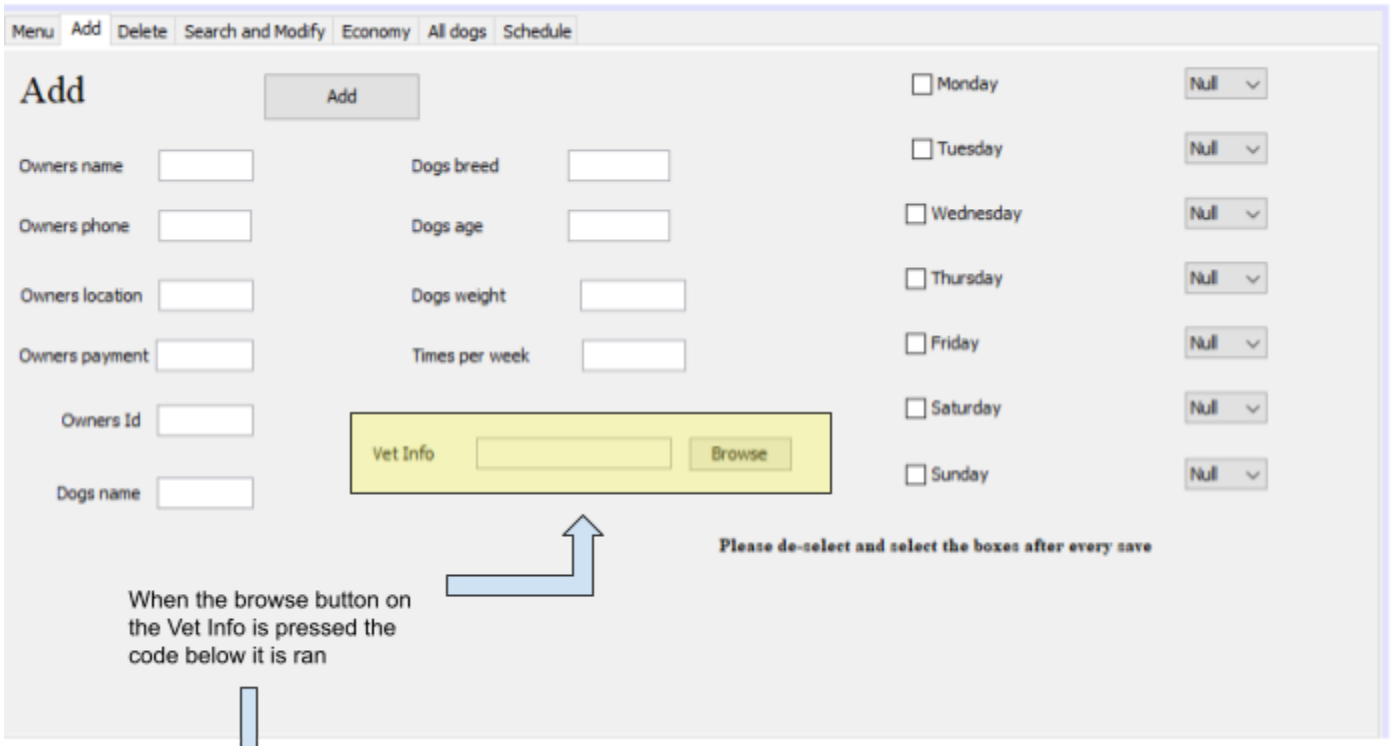
← FrontEnd (Visual)

List of Complex Techniques:

1) Use of additional libraries
2) Try/Catch
3) Complex Selection
4) Loops
5) File Handling
6) Encapsulation
7) GUI

1) Use of additional Libraries

During my code, I decided to use an additional library which was a JFileChooser. I chose to use this technique since with this I was able to save the pathway of the file found on the client's computer. This made it easier to find the other documents saved in this attribute. In my code when the user is on the add page, in the Vet info section a browse button is pressed and this opens a file explorer kind of tab where the user can select the file of his choosing. The pathway to that specific file is saved in the box beside it so that the user can confirm he picked the right one.



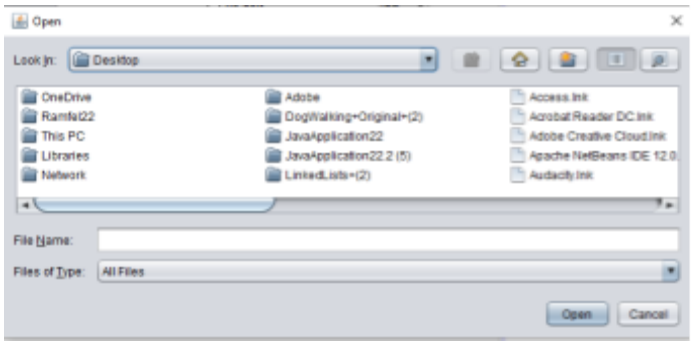

```
JFileChooser chooser = new JFileChooser();

chooser.showOpenDialog(null);

File f = chooser.getSelectedFile();
```

Here the FileChooser attribute is created

Here is where the file window is opened and the pathway of the file you chose is saved

```
if (f == null) {

    JOptionPane.showMessageDialog(this, "Please select file in the vet info location", "ERROR", JOptionPane.ERROR_MESSAGE);

    TxtImage.setText("");

} else {

    String filename = f.getAbsolutePath();

    TxtImage.setText(filename);

}
```

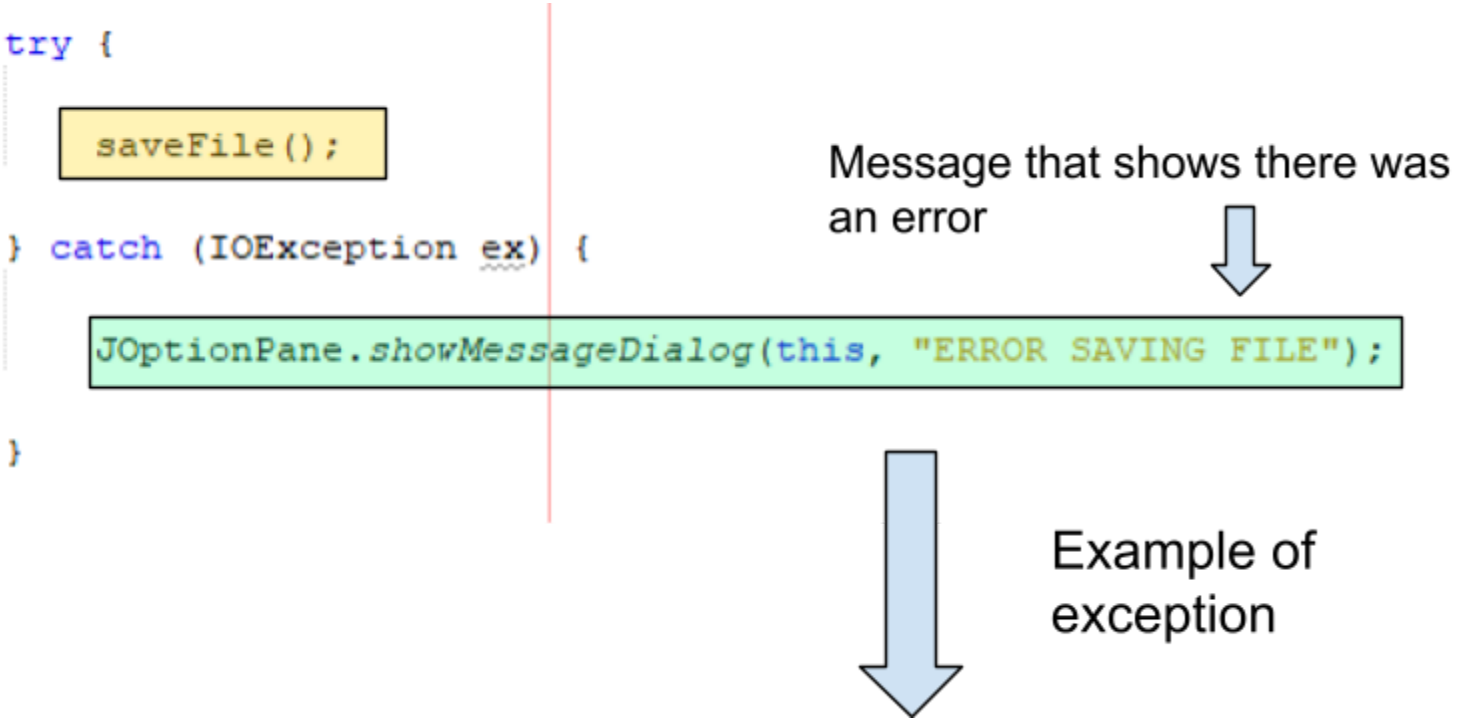
All this code is set up to react if the user leaves the window without choosing a file or an image it will show an error alerting him. If he selects something it is saves in the JTextField



This is the window that is displayed when the code is ran and the button is pressed

2) Try/Catch

During my code I used the Try/Catch to find those exceptions and catch them and let the program flow without any type of error while giving the user feedback so he can understand what is wrong. In this example, I used the Try/catch to detect if there was an issue while saving the file. If it encounters an error, it is caught by the Catch part and a message to the user is shown. An error message pops up, and the save file is not run.



```
try {
    line = read.readLine();
} catch (Exception e) {
    line = null;
```

3) Complex Selection

Here in order to verify that in the add section, there were no blank spaces in any of the multiple field texts. All the text fields are located and checked by the code; if any of them are blank, the message that there was an error pops up and terminates the add code before the dog can be saved.

```
if (name.equals("") || phone.equals("") || location.equals("") || id.equals("") || Dname.equals("") || breed.equals("")) {
    JOptionPane.showMessageDialog(this, "Enter no blank spaces please", "Error!", JOptionPane.ERROR_MESSAGE);
} else {
    Dog doggo = new Dog(name, phone, location, payment, id, Dname, breed, age, weight, times, vet);
    boolean found = dogsList.searchDuplicate(id);
    if (found == true) {
        JOptionPane.showMessageDialog(this, "Dog already created, please enter other variables", "Error!", JOptionPane.INFORMATION_MESSAGE);
```

The message that alert the user that his save failed

Another example of Complex selection would be here where I used nested Ifs in order to keep the code in one flowing motion, speeding up the process and making it much easier for the user

4) Loops

I used loops to check if there was a specific Dog on the list or a particular breed. Loops were mainly used to be able to find different elements and attributes in the various lists throughout my code. In this case, I used the while loop to go through my list of dogs and search quickly and efficiently if there is a dog with that specific Id or if there's just no dog at all. This If inside the loops is used to verify that the dog is either in the list or not. If the id matches the one given to us by the user, we save the information and return it to the front-end, while if it's not found, the else part of the if makes sure that you follow to the next node not to leave the whole loop in an infinite loop.

```
NodeDog aux = first;

while (aux != null) {
    if (aux.getData().getId().equals(id)) {
        return aux.getData();
    } else {
        aux = aux.getNext();
    }
}

return null;
}
```

The while starts using the auxiliary pointer as a base for decision making. Once it hits a null value the while will end

This part of the code is the one that goes line by line searching for the dog during the process to get the id in order to find the dog.

```
while (line != null) {

    String[] lineArray = line.split(",");

    String name = lineArray[0];

    String phone = lineArray[1];

    String location = lineArray[2];

    String p = lineArray[3];

    String id = lineArray[4];

    String dogName = lineArray[5];

    String breed = lineArray[6];

    String a = lineArray[7];

    String w = lineArray[8];

    String t = lineArray[9];

    String vet = lineArray[10];

    String mon = lineArray[11];
```

Another example where we use the while loop in order to go line by line saving the different dog´s attributes

5) File Handling

The use of file handling in my program allowed the user to freely exit the program and re-enter as many times as possible without worrying about losing all your processes and information. After successfully adding a dog to the list, a method to save the file called saveFile() is named and held right there and then. This makes it easier for the user to use and more reliable since it's an autosave. Then the text file where everything is stored is created; in my case, it is Doggie.Txt. Then the different variables get turned into strings and compacted into a single line of text to be saved in the Txt file. The reading of the text file is almost the same. This code is run at the very beginning when the user opens the program. The reader finds the text file and goes
line by line, taking out each component and changing them to their original value. This is later saved in each of the corresponding objects.

```
try {
    saveFile();
} catch (IOException ex) {
    JOptionPane.showMessageDialog(this, "ERROR SAVING FILE");
}
```

The save file code is ran.

```
String line = ownerName + "," + phone + "," + location + "," + payment + "," + id + "," + dogName + "," + breed + "," + age
out.write(line);
out.newLine();
aux = aux.getNext();
au = au.getNext();
}
```

Here the line that will later appear on the text file.

The line is written into the text and a new line is printed out

```
private void readFile() {
    String message = "";
    int lineno = 0;
    BufferedReader read;
    String line;
    try {
        read = new BufferedReader(new FileReader("Doggie.txt"));
        try {
            line = read.readLine();
    } catch (Exception e) {
            line = null;
    }
    while (line != null) {
        String[] lineArray = line.split(",");
        String name = lineArray[0];
        String phone = lineArray[1];
        String location = lineArray[2];
        String p = lineArray[3];
```

Method

Here the file with the information is looked for and it begins reading line by line

Here the different component of the line that was saved begin to decompose into their original information

```
String sun = lineArray[17];
message = "Age wrong: " + lineno + ". " + line;
int age = Integer.parseInt(a);
message = "Payment wrong: " + lineno + ". " + line;
double payment = Double.parseDouble(p);
message = "Weight wrong: " + lineno + ". " + line;
double weight = Double.parseDouble(w);
message = "Times wrong: " + lineno + ". " + line;
int times = Integer.parseInt(t);
```

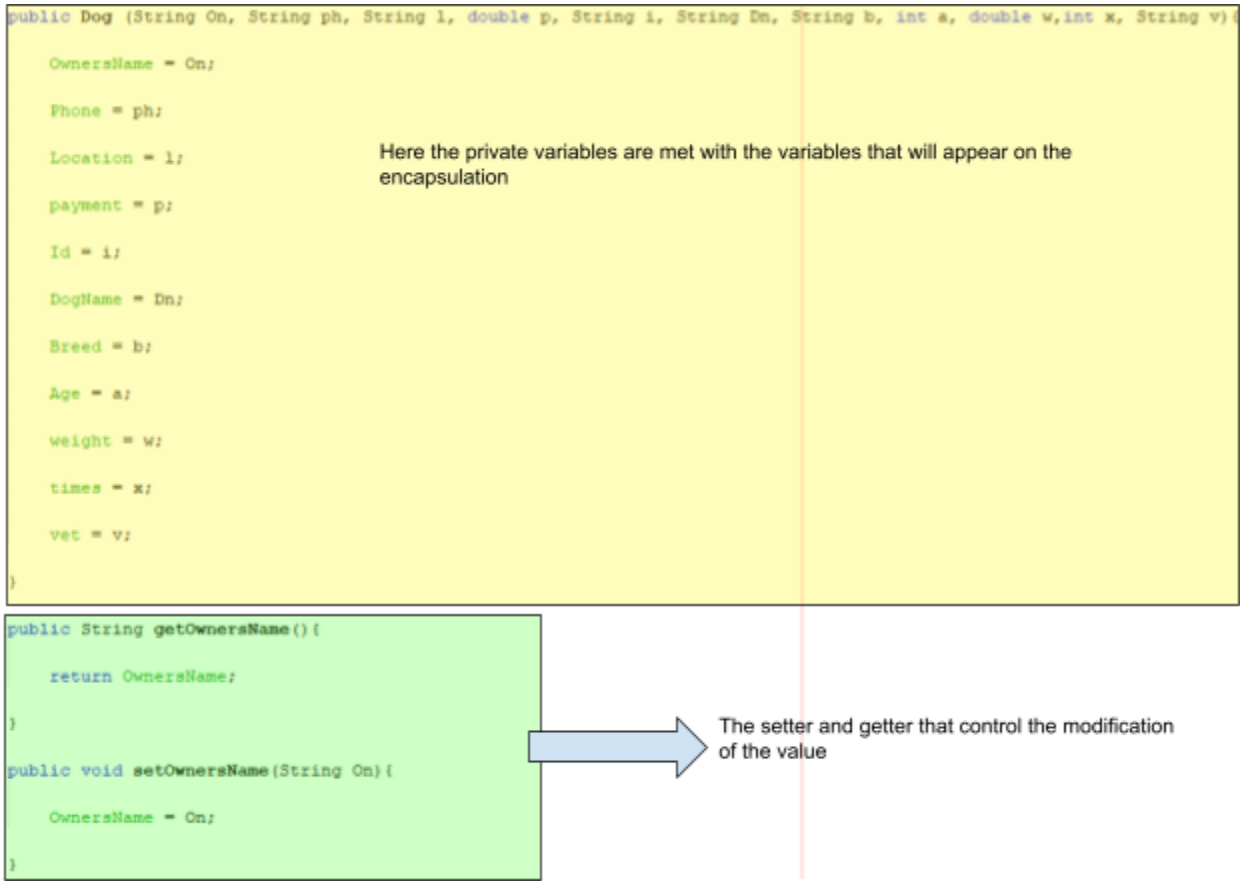
Here the variables are changed from String to their corresponding variables

```
message = "";
Dog doggo = new Dog(name, phone, location, payment, id, dogName, breed, age, weight, times, vet);
dogsList.addNode(doggo);
Schedule sc = new Schedule(location, dogName, mon, tue, wed, thu, fri, sat, sun);
scheduleList.addNode(sc);
try {
    line = read.readLine();
} catch (Exception e) {
    line = null;
}
}
```

Here the objects are created

6) Encapsulation

In my code I used encapsulation to keep them safe and to make it harder to be modified. In my code, I used the code in order to keep my variables and objects more secure. For instance, to control that the dog object does not tamper that easily I firstly created the variables in private and then later created the method on which the private variables would be called and the connection with the later setter and getters would occur. To modify the values of the variables I created both setter and getter and these helped me modify the value and get the value of private variables.

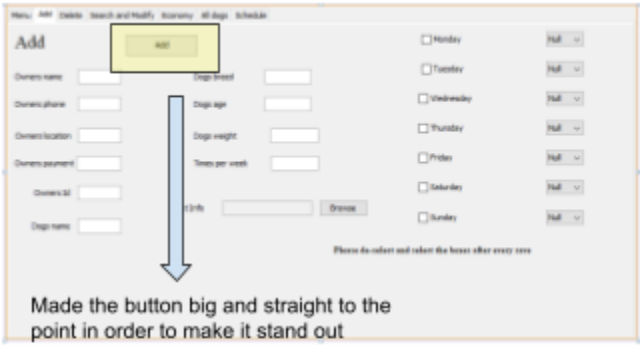


.

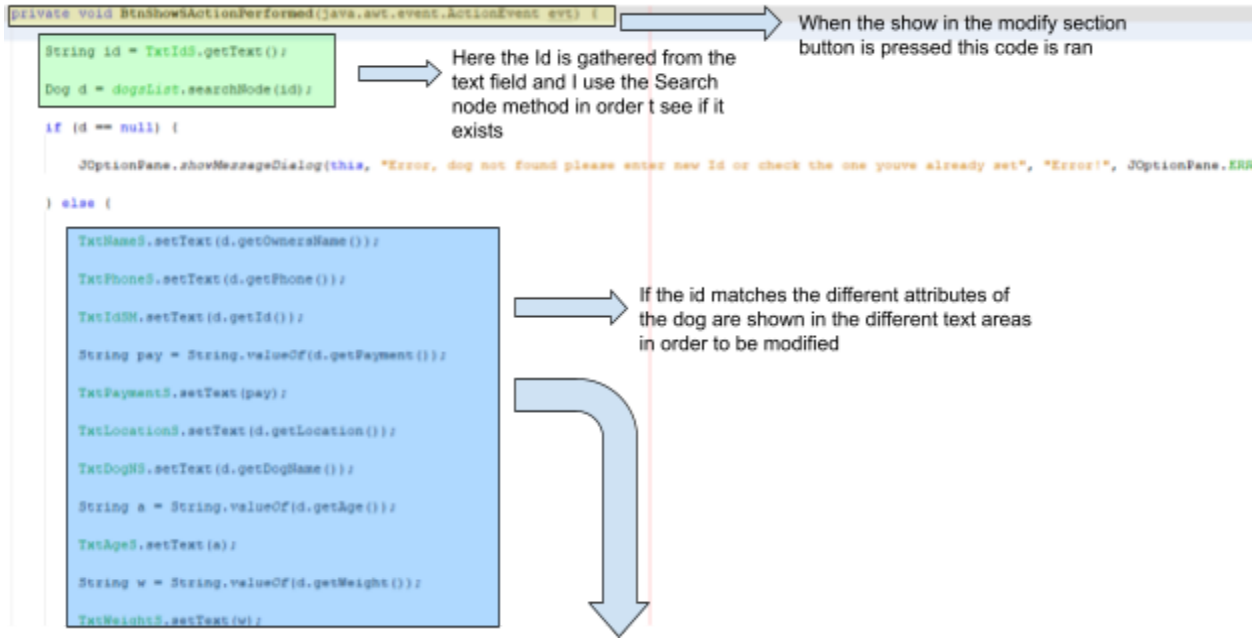
7) Graphic User Interface

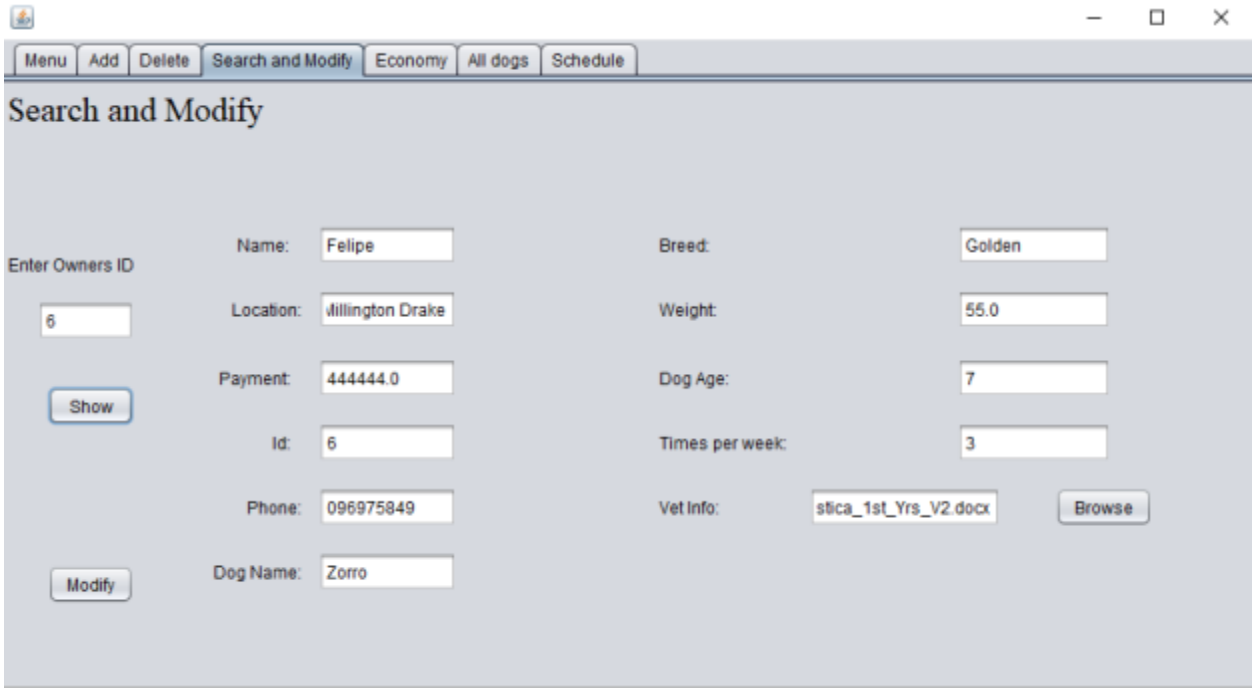
GUI makes data and information easier to process. For my GUI I decided that the use of Jtabbedpane was more efficient and more user-friendly than other methods. everything is correctly labeled beside its text field. All the buttons in my GUI are correctly labeled and are easy to notice and set apart from the other information. All the code for the buttons and text fields are coded accordingly and no text field has wandered off and not any of the buttons, text fields, and labels are useless or for decoration. Everything has a function.


Welcome
Bautista!
At the GUI I used JTabbedPane


Add
Made the button big and straight to the point in order to make it stand out


Search and Modify
Every text field is correctly labeled beside its corresponding label





Word count: 819