Assignment Cover Letter

(Individual Assignment)

Student Name:

Ardelia Shaula Araminta 2440065163

Course Code : COMP 6699

Course Name : Object Oriented Programming

Class : L2BC

Name of Lecturer :
Jude Martinez L

Major : Computer Science

Title of Assignment :
Something Fishy

Type of Assignment : Final Project

Due Date : 22 - 06 - 2021

Submission Date : 22 - 06 - 2021

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BINUS International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BINUS International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

Ardelia Shaula Araminta

Table of Contents

I) Introduction

It was a bit overwhelming in my second semester of Computer Science at BINUS International, and I spent most of my days learning and remembering topics given in class, but it's intriguing to recollect some of the lessons I've learned, even though I've studied Java. We were given a final project assignment at the end of the semester where students could explore and design their own program with specific requirements. Because I had so many ideas in my head, including learning JAVA Swing as my GUI, but due to the workload and time constraints, I prioritized and maximized the minimal criteria that needed to be met first. In the end, I decided to create an Aquatic Pet Clinic System called Something Fishy.

The project officially started on 21st May 2021. IntelliJ was chosen as my IDE for this project. Also, as this project's nature is open source, my initial commit was uploaded to my GitHub account, https://github.com/ardeliaraminta/OOPFinalProject/tree/main/src/oopfinal

Project Specification

There are many pet clinics to choose from, however I believe it is uncommon to locate facilities that accept ocean species such as sharks and whales. I chose this because I've always been interested in marine life, and sharks are my favorite animal. In this clinic, customers may check available schedules, register their pets, and make clinic appointments at Something Fishy, which means it allows both staff and customers to sign up and login. Staff, on the other hand, can manage their own profiles, appointments, and approvals or rejections of appointments. I believe this project has covered all the minimum requirements.

More specifications of the project:

Project Aim and purpose:

To create a fully functional clinic system that can be accessed by both customers and staff.

Project Audience:

Anyone interested in understanding how OOP works in Java, how to utilize Java File Handling (read and write files using java.io), and how to develop a basic yet complicated program, as well as those who are interested in marine life.

Project Requirements:

- Sign up and Login for Customer and Staff
- Approve and Decline Appointment (Staff)
- Manage staff profile (Staff)
- Book appointment and Schedule available (Customer)
- View Profile and Add new pet (Customer)

II) Solution Design

Main Window Menu (Main):
To select if the user is a member of the staff or a customer. Then they'll be directed to a page for either customers or employees. After that, you may choose to sign up as a new user or login if you've already registered.

Customer:

Menu (MainCostumer):

Choice of Login and Signup

Menu

1) Clinic Schedule
2) Add New Appointment
3) View Profile
4) Add Pet Data

1) Clinic Schedule: To see the schedule availability based on time and doctors available.
The customer is able to see the schedule and book appointment based on that.
2) View Appointment: To check the details of the appointment (Pet owner and pet details)
3) Add New Appointment: Appoint and book new appointment
4) View Profile: view costumer details
5) Add Pet Data: register a new pet (new to the clinic)

Staff:

- **Menu (MainStaff):**

Choice of Login and Sign up

- **Sign up:**

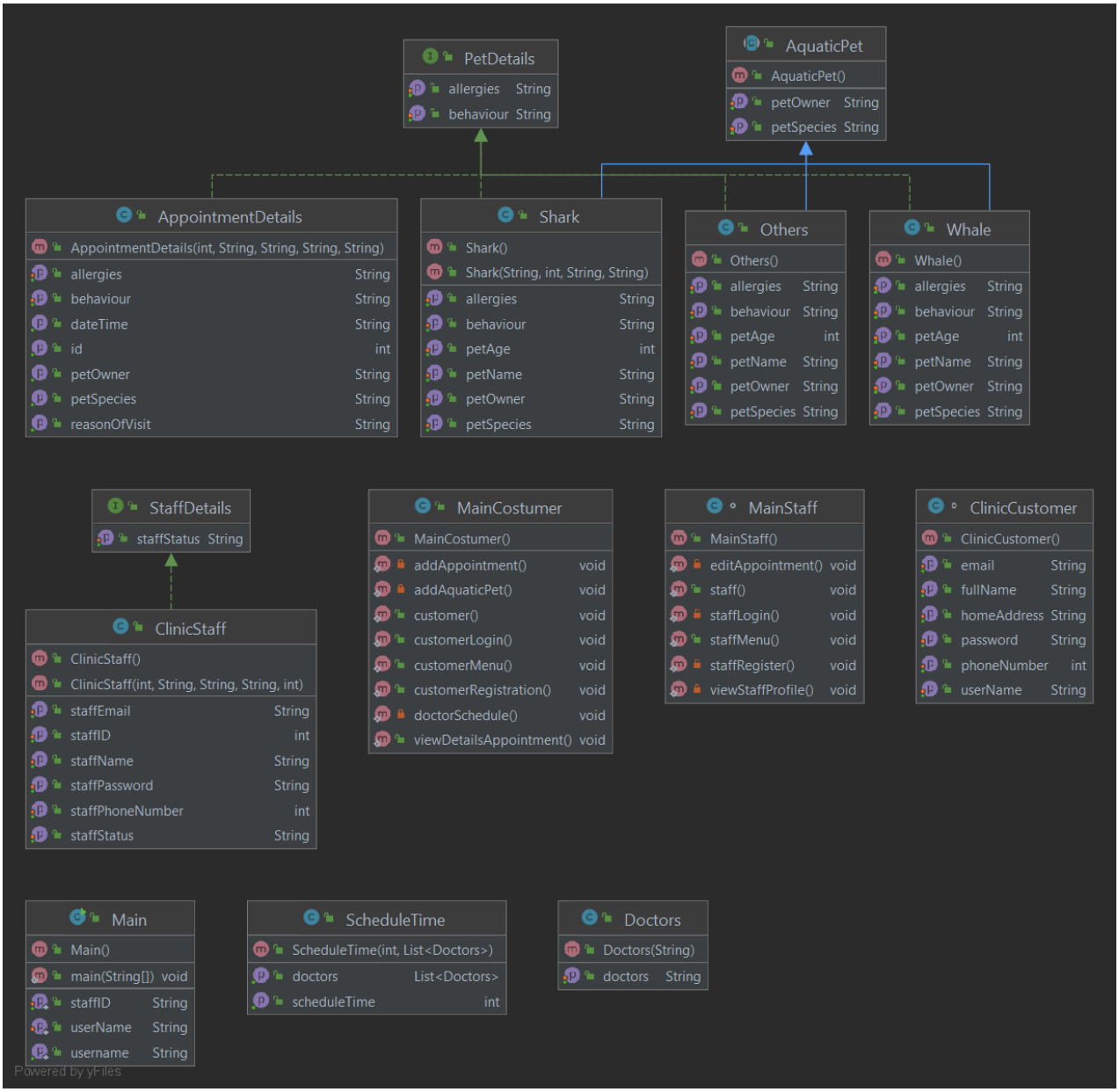
Entry: Username, Password, Email, Phone Number and Home Address

- **Login:**

For Customer: Entry of their username and password.

Menu
1) View User Profile
2) Cancel or Approve Appointment

III
UML DIAGRAM



IV) Solution Code

Interfaces

```
package oopfinal;


public interface PetDetails {
    /**
     *
     * @param behaviour
     */
    void setBehaviour(String behaviour);
    String getBehaviour();

    /**
     * @param allergies
     */
    void setAllergies(String allergies);
    String getAllergies();
}
```

<<interface>>

PetDetails

This particular interface has setter and getter for extra information of the animals such as behavior and allergies. This interface is used by, Whale, Shark, Others and Appointment Details classes.

```
package oopfinal;

public interface StaffDetails {
    void setStaffStatus(String status);
    String getStaffStatus();
}
```

<<interface>>

StaffDetails

This is another interface consisting getter setter for extra information of the staff whether or not they're working fulltime or part time.

1) Abstract Class

```
package oopfinal;

public abstract class AquaticPet {
    protected String behaviour;
    protected String allergies;
    String petOwner;
    String petSpecies;

    // constructor
    public AquaticPet() {}
```

```
    // getter & setter

    public abstract void setPetOwner(String petOwner);

    public abstract void setPetSpecies(String petSpecies);

    public abstract String getPetOwner();
    public abstract String getPetSpecies();

}
```

Abstract class: AquaticPet is the parent class as its getters setters are inherited to Whale, Shark, Others. It contains getter and setters of the pet owner and pet species.

AppointmentDetails implements Pet Details

```
package oopfinal;

public class AppointmentDetails implements PetDetails {

    private int id;
    private String petOwner;
    private String petSpecies;
    private String reasonOfVisit;
    private String dateTime;
    private String behaviour;
    private String allergies;


    public AppointmentDetails(int i, String own, String spec, String reason,
String date){
        id = i;
        petOwner = own;
        petSpecies = spec;
        reasonOfVisit = reason;
        dateTime = date;

    }

//setters
    public int getId(){
        return id;
    }

    public String getPetOwner(){
        return petOwner;
    }

    public String getPetSpecies(){
        return petSpecies;
    }

    public String getReasonOfVisit(){
```

```
        return reasonOfVisit;
    }

    public String getDateTime(){
        return dateTime;
    }

    @Override
    public void setBehaviour(String behaviour) {
        this.behaviour = behaviour;
    }

    @Override
    public String getBehaviour() {
        return behaviour;
    }

    @Override
    public void setAllergies(String allergies) {
        this.allergies = allergies;

    }

    @Override
    public String getAllergies() {
        return allergies;
    }
}
```

Clinic Staff: Class with details of staff

```
package oopfinal;

public class ClinicStaff implements StaffDetails {
    private String StaffName;
    private String StaffEmail;
    private String StaffPassword;
    private int StaffID;
    private int StaffPhoneNumber;
    private String status;

    public ClinicStaff() {
    }

    public ClinicStaff(int id, String name, String mail, String pass, int
phone) {
        StaffID = id;
        StaffName = name;
        StaffEmail = mail;
        StaffPassword = pass;
        StaffPhoneNumber = phone;
    }
```

```
    //getters setters

    public void setStaffName(String sName) {
        StaffName = sName;
    }

    public void setStaffEmail(String sEmail) {
        StaffEmail = sEmail;
    }

    public void setStaffPassword(String sPassword) {
        StaffPassword = sPassword;
    }

    public void setStaffID(int id) {
        StaffID = id;
    }

    public void setStaffPhoneNumber(int phone) {
        StaffPhoneNumber = phone;
    }

    public String getStaffName() {
        return StaffName;
    }

    public String getStaffEmail() {
        return StaffEmail;
    }

    public String getStaffPassword() {
        return StaffPassword;
    }

    public int getStaffID() {
        return StaffID;
    }

    public int getStaffPhoneNumber() {
        return StaffPhoneNumber;
    }

    @Override
    public void setStaffStatus(String status) {
        this.status = status;
    }

    @Override
    public String getStaffStatus() {
        return status;
    }
}
```

ClinicCustomer : Class with the details of the customer

```java
package oopfinal;

class ClinicCustomer {
    private String userName,password, fullName, email, homeAddress;
    private int phoneNumber;
    private AquaticPet pets;

    public ClinicCustomer(){
    };

    // assign values to details of Customer
    public void setUserName(String usr){
        userName = usr;
    }

    public void setPassword(String pwd){
        password = pwd;
    }

    public void setFullName(String fName){
        fullName = fName;
    }

    public void setEmail(String mail) {
        email = mail;
    }

    public void setPhoneNumber(int phone){
        phoneNumber = phone;
    }

    public void setHomeAddress(String add){
        homeAddress = add;
    }

    public void setPets(AquaticPet pet) {
        pets = pet;
    }

    // getters of details of the Customer

    public String getUserName(){
        return userName;
    }

    public String getPassword(){
        return password;
    }

    public String getFullName(){
        return fullName;
    }

    public String getEmail(){
        return email;
    }
```

```
    public int getPhoneNumber() {
        return phoneNumber;
    }

    public String getHomeAddress() {
        return homeAddress;
    }

    public AquaticPet getPets(){
        return pets;
    }

}
```

Doctors

```
package oopfinal;

public class Doctors {
        public String doctor;


        public Doctors(String docs){
            doctor = docs;
        }

        public void setDoctors(String doctor) {
            this.doctor = doctor;
        }

        public String getDoctors() {
            return doctor;
        }
    }
```

MainCostumer

This class is for only the customer to access.

Register and login into their account. First, they are asked to choose whether they want to login with their existing account or to register.

-

```
package oopfinal;

import java.io.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class MainCostumer {
```

```
    // redirected to login or register page
    public static void customer() {
        Scanner in = new Scanner(System.in);
        System.out.println(" ------------------------------------");
        System.out.println("\n\t\t\Customer");
        System.out.println(" ------------------------------------");
        System.out.println("1) Login");
        System.out.println("2) Register");

        System.out.println();
        System.out.print("Enter Choice           : ");
        int choice = in.nextInt();

        switch (choice) {
            case 1 -> customerLogin();
            case 2 -> customerRegistration();
            default -> {
                System.out.println("Invalid Key!");
                customer();
            }
        }
    }

public static void customerLogin() {
    String username, password, line;
    Scanner scan = new Scanner(System.in);

    //customer login page: Username and Password entry
    System.out.println("\t_____________________________________");
    System.out.println("\t           LOGIN                     ");
    System.out.println("\t_____________________________________");
    System.out.println();
    System.out.println("\tUsername: ");
    username = scan.next();
    System.out.println("\tPassword: ");
    password = scan.next();

    try {
        //read the custLog.txt file
        BufferedReader br = new BufferedReader(new
FileReader("custLog.txt"));
        boolean validation = false;

        // while loop, if the line is not empty (txt)  and contains username
and password
        // it checks with the validation, and proceed to the customer menu

        while ((line = br.readLine()) != null) {
            if (line.contains(username) && line.contains(password)) {
                validation = true;
                Main.setUserName(username);
                customerMenu();
            }
        }
```

CustomerLogin ()

The application will read the text file containing the username and password in CustomerLogin (). Then it checks if the text file is empty, if it isn't, it enters a while loop to read the text file line by line, and if the line matches the username and password entered by the user on the command line (it returns true), it goes to the customerMenu (), otherwise if its false, it will say that that password it exits the loop.

```
        // if the password is false
        if (!validation) {
            System.out.println("Password entered is wrong. ");
        }

        br.close();
    }
    // if the data not found
    catch (IOException ex) {
        System.out.println("File Not Found");
    }

}
```

```
public static void customerRegistration() {
    try {

        // written in the text file
        Scanner sc = new Scanner(System.in);
        File cust = new File("customer.txt");
        File custLog = new File("custLog.txt");
        BufferedWriter custBW = new BufferedWriter(new
FileWriter("customer.txt", true));
        BufferedWriter logBW = new BufferedWriter(new
FileWriter("custLog.txt", true));
        ClinicCustomer customer = new ClinicCustomer();

        // if there is txt file already write on new line
        if (cust.createNewFile()) {
        } else {
            custBW.newLine();
        }
        if (custLog.createNewFile()) {
        } else {
            logBW.newLine();
        }
```

Customer Registration ()

This method allows the user to register their new data into the text file, using **BufferedWriter** that allows to write text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings. In this case, the data of the new customer. I created an if else statement, to check if there is no existing text file it will create a new one to store the data, if there is then it will be written there. Then, the user can fill their data for the registration such as Username, password, email, phone number, address, and then write the text file with the entries.

Bw.close () -> to close the BufferedWriter initialize previously

```
        // enter data of the customer
        System.out.println("Enter your user name: ");
        customer.setUserName(sc.nextLine());
        // password
        System.out.println("Password: ");
        customer.setPassword(sc.nextLine());
        // fullname
        System.out.println("Full name:");
        customer.setFullName(sc.nextLine());
        // email
        System.out.println("Email:");
        customer.setEmail(sc.nextLine());
        System.out.println("Enter your phone number: ");
        // phone number
        customer.setPhoneNumber(Integer.parseInt(sc.nextLine()));
        //address
        System.out.println("Enter your address: ");
        customer.setHomeAddress(sc.nextLine());

        // write in text file the above entries
        custBW.write(customer.getUserName() + "\t" + customer.getPassword() +
"\t" + customer.getFullName() + "\t" + customer.getEmail() + "\t" +
customer.getPhoneNumber() + "\t" + customer.getHomeAddress() + "\t");
        logBW.write(customer.getUserName() + "\t" + customer.getPassword());

        custBW.close();
        logBW.close();

        System.out.println("\n Done \n");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
public static void customerMenu() throws IOException {
    Scanner sc = new Scanner(System.in);

    System.out.println("              Menu              ");
    System.out.println("--------------------------------");
    System.out.println(" 1) Clinic Schedule");
```

```
    System.out.println(" 2) View Appointment");
    System.out.println(" 3) Add New Appointment");
    System.out.println(" 4) Add Pet Data");

    System.out.println();
    System.out.print("Enter Choice           : ");
    int options = sc.nextInt();

    switch (options) {
        case 1 -> doctorSchedule(); // see available doctors
        case 2 -> viewDetailsAppointment(); // details of appointment
        case 3 -> addAppointment(); // add new appointment
        case 4 -> addAquaticPet(); // register new customer pet
        default -> {
            System.out.println(" Invalid choice ");
            MainStaff.staff();
        }
    }

}
```

CustomerMenu ()

This is the menu that the user will encounter after registering and login to their account, so it is personalized for them to choose the service they need which in this case there are four; to be able to see the doctor schedule that are available on certain time, view the details appointment whether it is approved or not by the clinic, create a new appointment as well as registering their new pet or unregistered pet.

```
private static void doctorSchedule() throws IOException {

    // create an instance of the doctors
    Doctors doctor1 = new Doctors("Dr.Killjoy");
    Doctors doctor2 = new Doctors("Dr.Sage");
    Doctors doctor3 = new Doctors("Dr.Reyna");

    // make an arraylist of doctors available at certain time for checkup
schedule
    List<Doctors> doctor1000 = new ArrayList<>(Arrays.asList(doctor3,
doctor2, doctor2, doctor1, doctor1, doctor3));
    List<Doctors> doctor1100 = new ArrayList<>(Arrays.asList(doctor1,
doctor1, doctor2, doctor2, doctor1, doctor1));
    List<Doctors> doctor1300 = new ArrayList<>(Arrays.asList(doctor3,
doctor2, doctor2, doctor1, doctor1, doctor2));
    List<Doctors> doctor1600 = new ArrayList<>(Arrays.asList(doctor2,
doctor2, doctor2, doctor3, doctor1, doctor2));
    List<Doctors> doctor1800 = new ArrayList<>(Arrays.asList(doctor2,
doctor3, doctor2, doctor1, doctor1, doctor2));
    List<Doctors> doctor2000 = new ArrayList<>(Arrays.asList(doctor3,
doctor2, doctor2, doctor3, doctor1, doctor2));
    List<Doctors> doctor2200 = new ArrayList<>(Arrays.asList(doctor2,
doctor2, doctor2, doctor1, doctor1, doctor2));
```

Doctors Schedule ()

```
List<Doctors> doctor1000 = new ArrayList<>(Arrays.asList(doctor3,
doctor2, doctor2, doctor1, doctor1, doctor3));
```

This essentially takes a list of doctors and makes a replica of it because Array List is initialized with a size, it grows in size automatically when new elements are added to it. However, the size can expand if the collection grows or shrink if objects are removed from the collection. For instance, if we want to add a new doctor, all we have to do is initialize:

Doctors doctor4 = new Doctors("Dr.Phoenix")

```
    // make an arraylist of time
    List<ScheduleTime> time = new ArrayList<>(Arrays.asList(new
ScheduleTime(1000, doctor1000),
            new ScheduleTime(1200, doctor1100), new ScheduleTime(1400,
doctor1300), new ScheduleTime(1600, doctor1600),
            new ScheduleTime(1800, doctor1800), new ScheduleTime(2000,
doctor2000), new ScheduleTime(2200, doctor2200)));
```

This will create an ArrayList of schedule from the existing list. This will be implemented to show the schedule time table along with available doctor (Make an Arraylist of time).

```
    System.out.println("\t\t Schedule Timetable");
    System.out.println("\t\t ____________________");
    System.out.println();
    System.out.println("Time:\t |Monday\t |Tuesday\t |Wednesday\t |Thursday\t
|Friday\t |Saturday\t |Sunday");

    for (ScheduleTime scheduleTime : time) {
        StringBuilder line = new StringBuilder();
        // for loop for the schedule time:  obtain doctors available on that
specific time and display on the days
        for (int j = 0; j < scheduleTime.getDoctors().size(); j++) {
            if (j != (scheduleTime.getDoctors().size() - 1)) {

line.append(scheduleTime.getDoctors().get(j).getDoctors()).append("\t |");
            } else {
                line.append(scheduleTime.getDoctors().get(j).getDoctors());
            }
}
```

Display the Schedule of the Available Appointment
String Builder - > append methods

For loop for the schedule time: to obtain doctors available on that specific time and displays on the days. This can be done by appending the time and the dr. (name) based on the Arraylist created before. After that, the customer can choose other services after seeing the available schedule for appointment.

```
// monday close
        System.out.println(scheduleTime.getScheduleTime() + "\t |Close \t |"
+ line);
    }

    customerMenu();

private static void addAppointment() throws IOException {
    // initiate the first id of the appointment
    LineNumberReader reader = null;
    int id = 0;
    try {
        // create a new file for appointment details
        Scanner scan = new Scanner(System.in);
        File appointment = new File("Appointment.txt");
        BufferedWriter appointmentBW = new BufferedWriter(new
FileWriter("appointment.txt", true));
        BufferedReader appointmentBR = new BufferedReader(new
FileReader("appointment.txt"));

        // if there is no file to be overwritten then make a new file else
add a new line or create a new line.
        if (appointment.createNewFile()) {
        } else {
            appointmentBW.newLine();
        }

        reader = new LineNumberReader(new FileReader("appointment.txt"));
        // Read file till the end
        while ((reader.readLine()) != null) {
            id = reader.getLineNumber() + 1;
        }
```

addAppointment ()

This method allows the program to create a new text file or read the existing the text file to write the details of the appointment. if there is no file to be overwritten then make a new file else add a new line or create a new line.

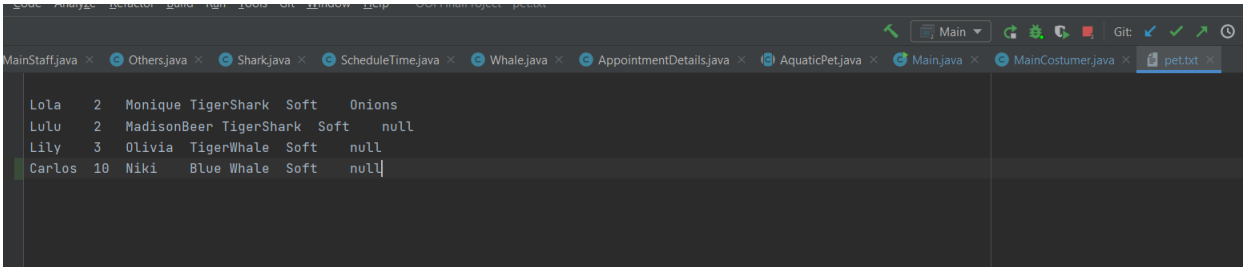
```
// read the pet.txt file to get information and details of the pet
String line = "invalid";
```

```
String petName = "invalid";
String petAllergies = "invalid";
String petSpecies = "invalid";
BufferedReader br = new BufferedReader(new FileReader("pet.txt"));
```

To decrease redundancy of data entered and increase efficiency, the customer doesn't have to enter the pet details all over again such as pet name, species or allergies but instead take the information from pet.txt.

```
// while the line is not empty and it contains the username, split each of
words and
// according to the index assign the species and pet name

while ((line = br.readLine()) != null) {
    if (line.contains(Main.getUsername())) {
        String[] text = line.split("\t");
```

This particular while loop allows the program to read per lines and split the words one by one so that the pet information can be accessed based on their index. In this case, looking from the text file as shown below:



Pet Name is on [0], Pet species is on the [3] index, and allergies [5] index
For example: Lola -> Name, Tiger Shark -> Species and Lola is allergic to onions.
The values can then be assigned to the text file. As a result, the customer only needs to input their appointment date, time, and the reason why their pet is being admitted at the clinic. The rest will be produced automatically.

```
// assign values taken from pet.txt based on index
        petSpecies = text[3];
        petName = text[0];
        petAllergies = text[5];
    }
}

br.close();
     //appointment details
```

```
        System.out.println("Appointment id: " + id);
        System.out.println("Enter Appointment Date (dd/mm/yyyy): ");
        String date = scan.nextLine();
        System.out.println("Enter Appointment Time (exp: 1430): ");
        String time = scan.nextLine();
        System.out.println("Username: " + Main.getUsername());
        String name = Main.getUsername();
        System.out.println("Pet Species: " + petSpecies);
        String petS = petSpecies;
        System.out.println("Pet Name: " + petName);
        String petN = petName;
        System.out.println("Pet Allergies: " + petAllergies);
        String petA = petAllergies;
        System.out.println("Appointment Reason: ");
        String reason = scan.nextLine();

public static void viewDetailsAppointment() {
    try {
        Scanner myReader = new Scanner(new File("appointment.txt"));

        // if there is data, print the following
        if (myReader.hasNextLine()) {
            System.out.println("\n\nAppointment Data\n");
            System.out.println("ID\tPet Name\tDate\t\t\tTime\t\tPet
Owner\t\tPet Species\t\tReason Visit \t\tApproval");

System.out.println("_____
_____");
        } else {
            System.out.println("\n\nNo appointment is found.\n");
        }

        // while there is line, split the words by tabs
        while (myReader.hasNextLine()) {
            String data = myReader.nextLine();
            String[] text = data.split("\t");

            // if there is data of the user create a new instance
            if (data.contains(Main.getUsername())) {
```

```
intment.txt   Main   Git:
Whale.java | AppointmentDetails.java | AquaticPet.java | Main.java | MainCostumer.java | appointment.txt

1
2
3   2   22/06/2021  1800    Monique Lola    Tiger Shark Sick    Approve
4
5   4   29/06/2021  1600    MadisonBeer Lulu    TigerShark  Sick    Approve
6
7   6   22/06/2021  1900    Olivia  Lily    Tiger Whale Sickness
8   8   11/07/2021  0900    Niki    Carlos  Blue Whale  Fever
```

ViewDetaisAppointment ()

Like previously implemented, I initiate a text file reader that will be able to read the text file and print out the appointment details from the appountment.txt and whether it is approved or not by the staff. To print out each of the details, the words are split and can be accessed by their indexes.

```
                AppointmentDetails appoint = new
AppointmentDetails(Integer.parseInt(text[0]), text[1], text[2], text[3],
text[4]);

                // if the words length is 7, get data based on the index
                if (text.length == 7) {
                    System.out.println(appoint.getId() + "\t" +
appoint.getDateTime() + "\t\t" + appoint.getPetOwner() + "\t\t" +
appoint.getPetSpecies() + "\t\t" + appoint.getReasonOfVisit()+ "\t\t" +
text[5] + "\t\t");
                } else {
                    System.out.println(appoint.getId() + "\t" +
appoint.getDateTime() + "\t\t" + appoint.getPetOwner() + "\t\t" + "\t\t" +
appoint.getPetSpecies() + "\t\t" + appoint.getReasonOfVisit()  + "\t\t" +
text[5] + "\t\t" + text[6] +"\t\t" + text[7]);
                }
            }
        }
        System.out.println("\n\n");
        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}/
```

Note: In every class method, I used to try catch for error handling.

```
private static void addAquaticPet() throws IOException {
        Scanner scan = new Scanner(System.in);

        try {
            // write in text file
            File pet = new File("pet.txt");
            BufferedWriter petBW = new BufferedWriter(new
FileWriter("pet.txt", true));

            // if there is no text file create new, if there is existing file
write a new line
            if (pet.createNewFile()) {
            } else {
                petBW.newLine();
            }
```

Add AquaticPet ()

This method is for the customer to register, write and add new unregistered pet to the clinic, I also use if-else statement to check whether there is a need to create a new text file or not.
The user can choose default pet species, which is the common pets to be admitted which are sharks and whales or they have different type user can choose others.

```
// choose pet species
System.out.println("Please choose yours pet species: ");
System.out.println("1) Whale");
System.out.println("2) Shark");
System.out.println("3) Others");

// choose 1/2/3
System.out.println();
System.out.print("Enter Choice           : ");
int opt = scan.nextInt();
scan.nextLine();
```

After they choose, the program will go into a switch case statement. Each case statement will go into the specific species they choose and create a new instance of the species, which then will be stored into a text file of pet.txt.

```
            // switch statement for the selected pet species
            //write in text file the details
            //case 3 for other species -> Others
            switch (opt) {
                case 1 -> {
                    Whale whale = new Whale();
                    System.out.println("Enter your Pet name: ");
                    whale.setPetName(scan.nextLine());
                    System.out.println("Enter your Pet age: ");
                    whale.setPetAge(Integer.parseInt(scan.nextLine()));
                    whale.setPetOwner(Main.getUsername());
                    System.out.println("Enter your Pet species: ");
                    whale.setPetSpecies(scan.nextLine() + " Whale");
                    System.out.println("Pet Behaviour; Aggressive/ Soft ");
                    whale.setBehaviour(scan.nextLine());
                    System.out.println("Any particular allergies: ");
                    whale.setAllergies(scan.nextLine());
                    petBW.write(whale.getPetName() + "\t" + whale.getPetAge()
+ "\t" + whale.getPetOwner() + "\t" + whale.getPetSpecies() + "\t" +
whale.getBehaviour() + "\t" + whale.getAllergies());
                }
                case 2 -> {
                    Shark shark = new Shark();
                    System.out.println("Enter your Pet name: ");
                    shark.setPetName(scan.nextLine());
                    System.out.println("Enter your Pet age: ");
                    shark.setPetAge(Integer.parseInt(scan.nextLine()));
                    shark.setPetOwner(Main.getUsername());
                    System.out.println("Enter your Pet species: ");
```

```
                    shark.setPetSpecies(scan.nextLine() + " Shark");
                    System.out.println("Pet Behaviour; Aggressive/ Soft ");
                    shark.setBehaviour(scan.nextLine());
                    System.out.println("Any particular allergies: ");
                    shark.setAllergies(scan.nextLine());
                    petBW.write(shark.getPetName() + "\t" + shark.getPetAge()
+ "\t" + shark.getPetOwner() + "\t" + shark.getPetSpecies() + "\t" +
shark.getBehaviour() + "\t" + shark.getAllergies());
                }
                case 3 -> {
                    Others other = new Others();
                    System.out.println("Enter your Pet name: ");
                    other.setPetName(scan.nextLine());
                    System.out.println("Enter your Pet age: ");
                    other.setPetAge(Integer.parseInt(scan.nextLine()));
                    other.setPetOwner(Main.getUsername());
                    System.out.println("Enter your Pet species: ");
                    other.setPetSpecies(scan.nextLine());
                    System.out.println("Pet Behaviour; Aggressive/ Soft ");
                    other.setBehaviour(scan.nextLine());
                    System.out.println("Any particular allergies: ");
                    other.setAllergies(scan.nextLine());
                    petBW.write(other.getPetName() + "\t" + other.getPetAge()
+ "\t" + other.getPetOwner() + "\t" + other.getPetSpecies() + "\t" +
other.getBehaviour() + "\t" + other.getAllergies());
                }
                default -> {
                    System.out.println("Invalid");
                    MainStaff.staff();
                }
            }

            petBW.close();
            System.out.println("\n  Done! \n");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
        customerMenu();
    }
}
```

Why do I do this?

So that the user can automatically generate the species without the need to worry whether they enter wrong information or pet species. After they're done, the user will be directed to the customer menu in which they can add new appointment or do other services available.

Clinic Staff

```
package oopfinal;
import java.io.*;
import java.util.Scanner;
```

```
class MainStaff {
    private static Scanner input;

    public static void staff() {
        // staff main menu ( register or login )
        input = new Scanner(System.in);
        System.out.println(" ------------------------------------");
        System.out.println("\n\t\t\tStaff");
        System.out.println(" ------------------------------------");
        System.out.println("1)Register");
        System.out.println("2)Login");

        System.out.println();
        System.out.println("Enter Choice   : ");
        int option = input.nextInt();

        switch (option) {
            case 1 -> staffRegister();
            case 2 -> staffLogin();
            default -> {
                System.out.println("invalid choice !");
                staff();
            }
        }
    }
```

```
//new staff register
private static void staffRegister() {
    try {
        // create a file for the details of teh staff and staff verification
        Scanner scan = new Scanner(System.in);
        File staff = new File("staff.txt");
        File custlog = new File("staffLog.txt");
        BufferedWriter staffBW = new BufferedWriter(new
FileWriter("staff.txt", true));
        BufferedWriter logbw = new BufferedWriter(new
FileWriter("staffLog.txt", true));
        ClinicStaff st = new ClinicStaff();

        if (staff.createNewFile()) {
        } else {
            staffBW.newLine();
        }
        if (custlog.createNewFile()) {
        } else {
            logbw.newLine();
        }

        // enter staff details
        System.out.println("Enter your ID: ");
        st.setStaffID(Integer.parseInt(scan.nextLine()));
        System.out.println("Enter your Name: ");
        st.setStaffName(scan.nextLine());
        System.out.println("Enter your Email: ");
        st.setStaffEmail(scan.nextLine());
        System.out.println("Enter your Password: ");
```

```
        st.setStaffPassword(scan.nextLine());
        System.out.println("Enter your phone number: ");
        st.setStaffPhoneNumber(Integer.parseInt(scan.nextLine()));
        System.out.println("Status: Full-Time / Part-Time");
        st.setStaffStatus(scan.nextLine());

        // write in the txt
        staffBW.write(st.getStaffID() + "\t" + st.getStaffName() + "\t" +
st.getStaffEmail() + "\t" + st.getStaffPassword() + "\t" +
st.getStaffPhoneNumber()+ "\t" + st.getStaffStatus());
        logbw.write(st.getStaffID() + "\t" + st.getStaffPassword());
        staffBW.close();
        logbw.close();

        System.out.println("\nSuccessfully wrote to the file.\n");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

StaffRegistration():

This class is similar with MainCostumer () for registration of new account. This will take the information from the staff and store it in a text file called staff.txt and staff.Log.

```
private static void staffLogin() {
    String password, line;
    String id;

    System.out.println("\t_____________________________________");
    System.out.println("\t|          | LOGIN |                |");
    System.out.println("\t_____________________________________");
    System.out.println();
    System.out.println("\tEnter your Staff ID: ");
    id = input.next();
    System.out.println("\tEnter your Password: ");
    password = input.next();




}
```

Staff Login ()

This program will read a file staffLog.txt for login verification, when the text file is not empty split text by tab and check if the first index = id entered or registered and second index = password, if the data matched it will then proceed to open staff menu. If it doesn't it will say "Wrong Password".

```
try {
    // to read a file staffLog for login verification
    BufferedReader br = new BufferedReader(new FileReader("staffLog.txt"));
    boolean authenticated = false;

    // when the text file is not empty split text by tab
    while ((line = br.readLine()) != null) {
        String[] text = line.split("\t");

        // if the first index = id entered or registered and second index =
password
        // proceed open staff menu
        if (text[0].equals(id) && text[1].equals(password)) {
            // if the data matched open staff menu
            authenticated = true;
            Main.setStaffID(id);
            staffMenu();
        }
    }
```

When the user is logging in after registering their new account, here I used Buffered Reader to read the staffLog.txt which is a text file containing the staff Id and password for easier login verification. The while loop first check whether there is a line, if there is it will split per word. Therefore, that makes id = index[0] and password index[1], with this, we can check if these match with the entered password and id, if it matches (using Boolean) it will proceed to the staffMenu() if the password not, it will declare wrong inputted password, and there is true catch exception if the file is not found.

```
        //if either doesn't match
        if (!authenticated) {
            System.out.println("Wrong Password");
        }

        br.close();
    } catch (IOException ex) {
        System.out.println("File Not Found");
    }

}
```

```
// option for the staff menu
public static void staffMenu() {

    System.out.println("              Menu              ");
    System.out.println("--------------------------------");
    System.out.println(" 1) View User Profile");
    System.out.println(" 2) Cancel or Approve Appointment");
```

```
    System.out.println();
    System.out.println(" Enter choice        :");
    int options = input.nextInt();

    switch (options) {
        case 1 -> viewStaffProfile(); // to see the user profile
        case 2 -> editAppointment(); // approve or decline incoming
appointment from the customer
        default -> {
            System.out.println("Invalid Key!");
            MainStaff.staff();
        }
    }
}
```

Staff Menu ()

In the Staff Menu (), the user can choose to view the staff profile or cancel or approve appointment. The user will be given two option to choose, to see the staff profile and to be able to cancel or approve appointment of the customers.

```
private static void editAppointment() {
    try {
        // scanner to read the file
        Scanner appoint = new Scanner(new File("appointment.txt"));
        StringBuilder buffer = new StringBuilder();
        Scanner scan = new Scanner(System.in);
        int count = 0;

        // if there is data read per lines and split by tabs
        while (appoint.hasNextLine()) {
            String data = appoint.nextLine();
            String[] text = data.split("\t");

            // if the text file consist of 8 words it allows the staff to
decline or accept the appointment
            // to check if the data required is available
            if (text.length == 8) {
                System.out.println(data);
                System.out.println("\nApprove/Reject: ");
                String status = scan.next();
                buffer.append(data).append("\t").append(status);
            } else {
                buffer.append(data);
            }
            count++;
            if (count > 0) {
                buffer.append("\n");
            }
        }
        FileOutputStream output = new FileOutputStream("appointment.txt");
        output.write(buffer.toString().getBytes());
        output.close();
        appoint.close();
```

```
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

edit Appointment ()

This program allows to read the appointment.txt and split the data by tabs so they can be accessed per index. if the text file consists of 8 words it allows the staff to decline or accept the appointment, to check if the data required is available. This can be achieved by opening the appointment.txt and append the text (Approve / Reject) as the 7th index value so when the customer opens the appointment, they can see whether the clinic approves or decline their booking requests. If the data is not equal to 7 words, it won't be able to do anything with the data.

```
    // to see the details of the staff
    private static void viewStaffProfile() {
        try {
            // to obtain the data
            Scanner myReader = new Scanner(new File("staff.txt"));

            // to split the words in the text file
            while (myReader.hasNextLine()) {
                String data = myReader.nextLine();
                String[] text = data.split("\t");

            // if there is data containing the staff id, get the password
        and encrypt it ***** ( confidential )
                if (data.contains(Main.getStaffID())) {
                    StringBuilder pwd = new StringBuilder();
                    for (int i = 0; i < text[1].length(); i++) {
                        pwd.append("*");
                    }

 ClinicStaff staff = new ClinicStaff(Integer.parseInt(text[0]), text[1],
text[2], text[3], Integer.parseInt(text[4]));
                    // print the staff details
                System.out.println("\n\n\tProfile Data\n");
                System.out.println("ID\t: " + staff.getStaffID());
                System.out.println("Name\t: " + staff.getStaffName());
                System.out.println("Email\t\t: " + staff.getStaffEmail());
                System.out.println("Phone Number\t: 0" +
staff.getStaffPhoneNumber());
                    System.out.println("Password\t: " + pwd);
                }
            }
            System.out.println("\n\n");
            myReader.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
```

```
            e.printStackTrace();
        }

    }
}
```

viewStaff Profile ()

In the method, the staff members can see other profile, to obtain the data, the program will read text file of statff.txt and split the words in the text file, if loop will check if the data contains the staff data it will print the details.

Public Class Others extends AquaticPet implements PetDetails

```
package oopfinal;

public class Others extends AquaticPet implements PetDetails  {

    String petName;
    int petAge;

    public Others(){

    };

    public void setPetName(String petName){
        this.petName = petName;
    }

    public void setPetAge(int petAge){
        this.petAge = petAge;
    }

    public String getPetName(){
        return petName;
    }

    public int getPetAge(){
        return petAge;
    }

    @Override
    public void setPetOwner(String petOwner) {
        this.petOwner = petOwner;

    }

    @Override
    public void setPetSpecies(String petSpecies) {
        this.petSpecies = petSpecies;

    }
```

```
    @Override
    public String getPetOwner() {
        return petOwner;
    }

    @Override
    public String getPetSpecies() {
        return petSpecies;
    }

    @Override
    public void setBehaviour(String behaviour) {
        this.behaviour = behaviour;
    }

    @Override
    public String getBehaviour() {
        return behaviour;
    }

    @Override
    public void setAllergies(String allergies) {
        this.allergies = allergies;

    }

    @Override
    public String getAllergies() {
        return allergies;
    }
}
```

Public class Schedule Time

```
package oopfinal;

import java.util.List;

public class ScheduleTime {
    private int availTime;
    List<Doctors> doctors;

    public ScheduleTime(int time, List<Doctors> doc) {
        availTime = time;
        doctors = doc;
    }

    public int getScheduleTime(){
        return availTime;
    }

    public List<Doctors> getDoctors(){
        return doctors;
```

```
    }

}
```

Public class Shark extends AquaticPet implements PetDetails

```
package oopfinal;

public class Shark extends AquaticPet implements PetDetails{
    String petName;
    int petAge;

    public Shark(){

    };

    // construct Shark
    public Shark(String petName, int petAge, String pOwner, String pSpec ){
        super.petOwner = pOwner;
        super.petSpecies = pSpec;
        this.petName = petName;
        this.petAge = petAge;

    }

    public void setPetName(String petName){
        this.petName = petName;
    }

    public void setPetAge(int petAge){
        this.petAge = petAge;
    }

    public String getPetName(){
        return petName;
    }

    public int getPetAge(){
        return petAge;
    }

    @Override
    public void setPetOwner(String petOwner) {
        this.petOwner = petOwner;

    }

    @Override
    public void setPetSpecies(String petSpecies) {
        this.petSpecies = petSpecies;

    }

    @Override
    public String getPetOwner() {
        return petOwner;
    }
```

```
    @Override
    public String getPetSpecies() {
        return petSpecies;
    }

    @Override
    public void setBehaviour(String behaviour) {
        this.behaviour = behaviour;

    }

    @Override
    public String getBehaviour() {
        return behaviour;
    }

    @Override
    public void setAllergies(String allergies) {
        this.allergies = allergies;

    }

    @Override
    public String getAllergies() {
        return allergies;
    }
}
package oopfinal;

public class Whale extends AquaticPet implements PetDetails{
    String petName;
    int petAge;

    public Whale(){

    };

    public void setPetName(String petName){
        this.petName = petName;
    }

    public void setPetAge(int petAge){
        this.petAge = petAge;
    }

    public String getPetName(){
        return petName;
    }

    public int getPetAge(){
        return petAge;
    }

    @Override
    public void setPetOwner(String petOwner) {
        this.petOwner = petOwner;
```

```
    }

    @Override
    public void setPetSpecies(String petSpecies) {
        this.petSpecies = petSpecies;

    }

    @Override
    public String getPetOwner() {
        return petOwner;
    }

    @Override
    public String getPetSpecies() {
        return petSpecies;
    }


    @Override
    public void setBehaviour(String behaviour) {
        this.behaviour = behaviour;

    }

    @Override
    public String getBehaviour() {
        return behaviour;
    }

    @Override
    public void setAllergies(String allergies) {

    }

    @Override
    public String getAllergies() {
        return allergies;
    }

}
```

public class Others extends AquaticPet implements PetDetails

```
package oopfinal;

public class Others extends AquaticPet implements PetDetails  {

    String petName;
    int petAge;

    public Others(){

    };

    public void setPetName(String petName){
        this.petName = petName;
```

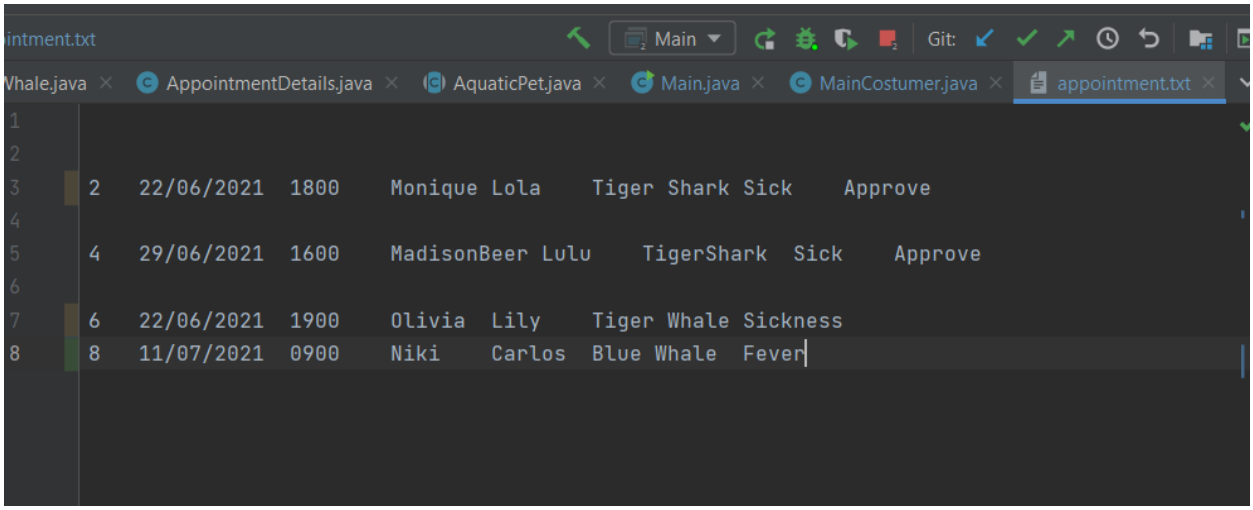
```
    }

    public void setPetAge(int petAge){
        this.petAge = petAge;
    }

    public String getPetName(){
        return petName;
    }

    public int getPetAge(){
        return petAge;
    }

    @Override
    public void setPetOwner(String petOwner) {
        this.petOwner = petOwner;

    }

    @Override
    public void setPetSpecies(String petSpecies) {
        this.petSpecies = petSpecies;

    }

    @Override
    public String getPetOwner() {
        return petOwner;
    }

    @Override
    public String getPetSpecies() {
        return petSpecies;
    }

    @Override
    public void setBehaviour(String behaviour) {
        this.behaviour = behaviour;
    }

    @Override
    public String getBehaviour() {
        return behaviour;
    }

    @Override
    public void setAllergies(String allergies) {
        this.allergies = allergies;

    }

    @Override
    public String getAllergies() {
        return allergies;
    }
}
```

Main.java

```
package oopfinal;

import java.util.Scanner;

public class Main {

    private static Scanner Line;
    private static String user;

    public static void setStaffID(String us){
        user = us;
    }
    public static String getStaffID() {
        return user;
    }


    public static void setUserName(String us) {
        user = us;
    }

    public static String getUsername() {
        return user;
    }

    public static void main ( String[] args){

        Line = new Scanner(System.in);

        boolean cont = false;
        while ( !cont) {
            Scanner ent = new Scanner(System.in);

            System.out.println(" ------------------------------------------
-----------------");
            System.out.println("|
|");
            System.out.println("| Welcome To Something Fishy: Ocean Animal
Pet Center System  |");
            System.out.println("|
|");
            System.out.println(" ------------------------------------------
----------------- ");
            System.out.println("|                          A) Staff
|");
            System.out.println("|                          B) Customer
|");
            System.out.println("------------------------------------------
------------------");

            System.out.println();
            System.out.println("Select your choice:           ");
            String choice = ent.next();
```

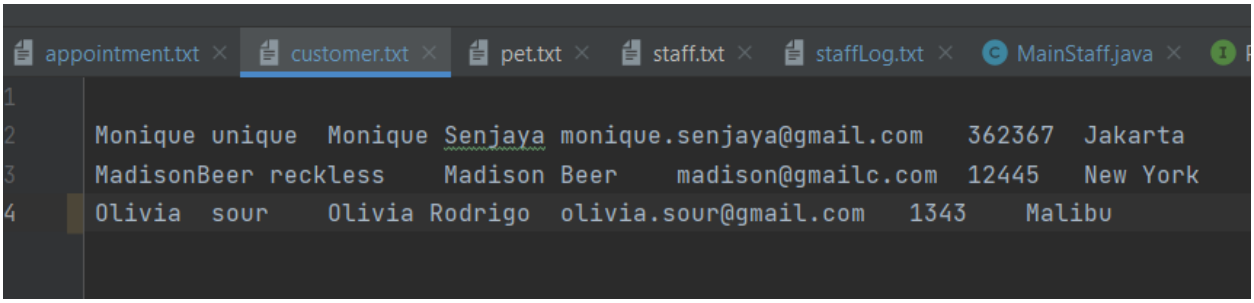
```
            // direct them to customer or staff main
            if (choice.toUpperCase().charAt(0) == 'A'){
                MainStaff.staff();
            } else if (choice.toUpperCase().charAt(0) == 'B'){
                MainCostumer.customer();
            } else {
                System.out.println("Invalid Choice!");
                cont = false;
            }

            new Main();


        }

    }
}
```

V) Example of Existing Text Files:



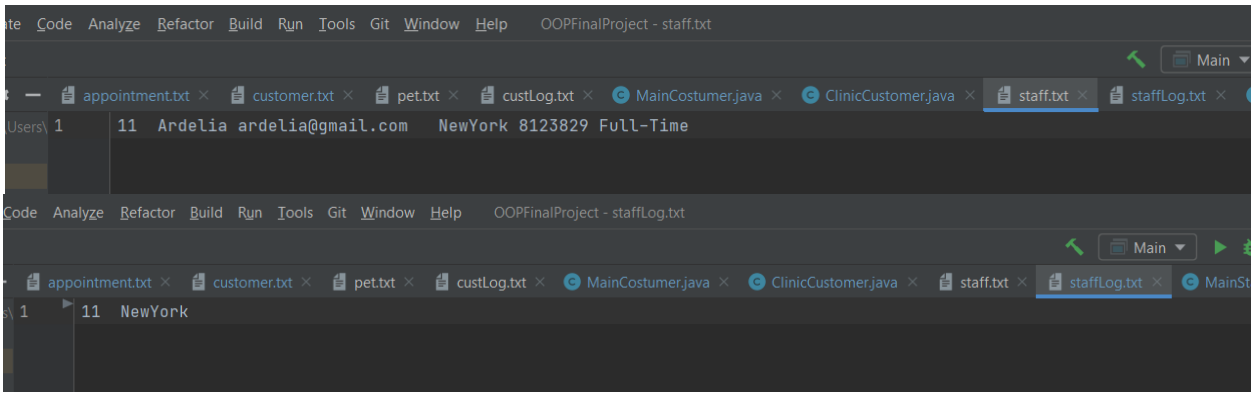
Appointment.txt

This is for the details of the appointment, both staff and customer can access this, staff can approve or reject the appointment, while customer can add new appointment and see whether the clinic reject or approve it.



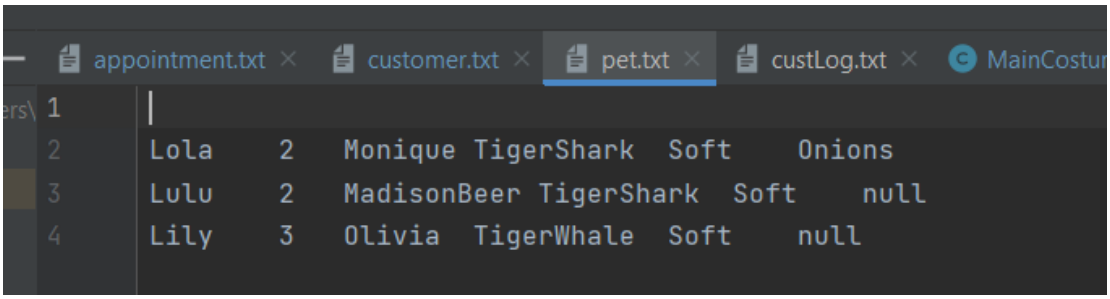
Customer.txt and CustLog.txt

This contains the data of each customer that registered put on Customer.txt and the customer login details can be accessed from CustLog.txt.



staff.txt and staffLog.txt

This contains the data of each staff that registered put on staff.txt and the customer login details can be accessed from staffLog.txt.



pet.txt

This text file contains the details of the pet registered.

VI) Screenshots of Working Program

a)

```
exclude patterns:
 ------------------------------------------------------------
|                                                            |
| Welcome To Something Fishy: Ocean Animal Pet Center System |
|                                                            |
 ------------------------------------------------------------
|                       A) Staff                             |
|                       B) Customer                          |
------------------------------------------------------------

Select your choice:
```

b)

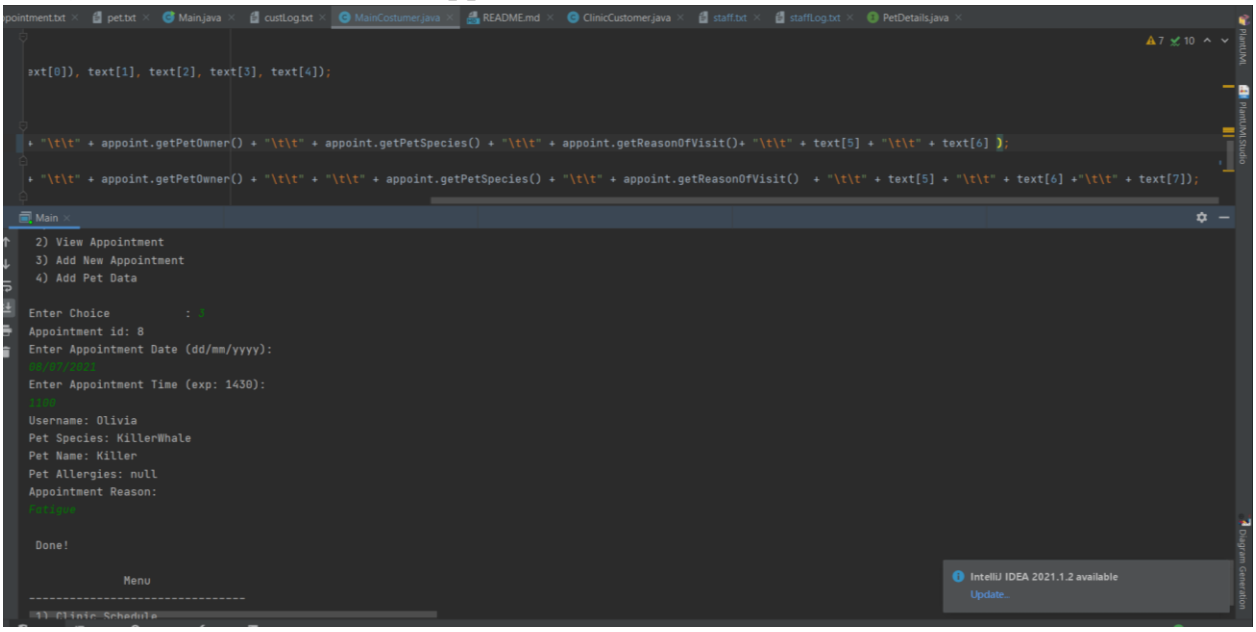
```
 ----------------------------------

          Staff
 ----------------------------------
1)Register
2)Login

Enter Choice   :
1
Enter your ID:
20
Enter your Name:
Ariana
Enter your Email:
ariana@gmail.com
Enter your Password:
MadisonSquare
Enter your phone number:
15634
Status: Full-Time / Part-Time
Full-Time

Successfully wrote to the file.

 ------------------------------------------------------------
```

a) Main Menu
b) Register Account Staff Member

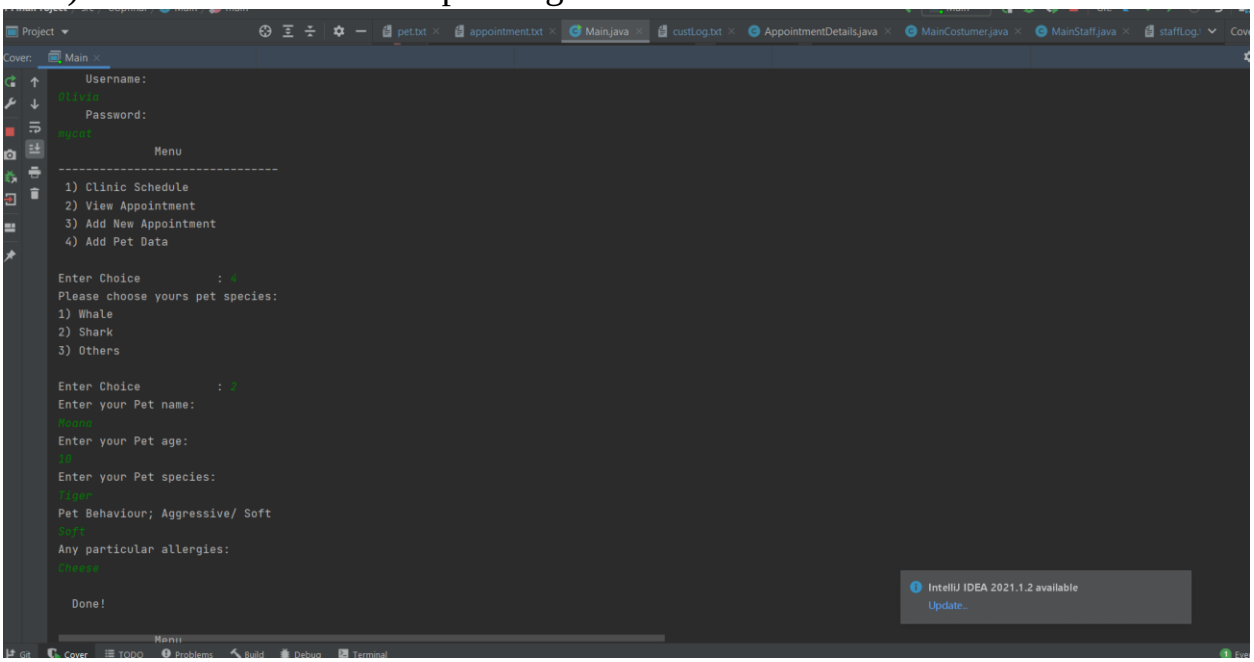
c) Login Staff and View Staff Profiles

```
Enter Choice   :
2
    ----------------------------------
    |          | LOGIN |             |
    ----------------------------------

    Enter your Staff ID:
20
    Enter your Password:
MadisonSquare
              Menu
-------------------------------
 1) View User Profile
 2) Cancel or Approve Appointment

 Enter choice        :
1


    Profile Data

ID  : 20
Name    : Ariana
Email       : ariana@gmail.com
Phone Number    : 015634
Password    : ******
```

d) Customer -> add new appointment



e) Customer -> Add new pet / register



f) Staff Approve/ Decline appointment