Object Oriented Programming
Project Document
MediMeet (Doctor Appointment System)
SUBMITTED BY:

Hasaan Ahmad	SP22-BSE-017
Muhammad Mujtaba	SP22-BSE-036
Muhammad Haider Sheikh	SP22-BSE-033

Supervisor: Sir Muzaffar Iqbal

Doctor Appointment Management System

Description:

The Doctor Appointment Management System is a comprehensive and user-friendly Java app designed to streamline and automate the process of managing appointments between doctors and patients. It provides a convenient platform for doctors, patients, and administrators to interact, schedule appointments, and maintain essential medical records.

Key Features:

1. User Registration and Login: The system allows doctors, patients, and administrators to register their profiles and securely log in using their credentials.

2. Doctor Dashboard: Doctors can access their personalized dashboard to manage their appointments,view patient details, and prescribe medications. They can also maintain their professional profiles, including specialization, qualifications etc.

3. Patient Dashboard: Patients have their own dashboard to schedule appointments with preferred doctors, view upcoming and past appointments and access prescriptions, they also have the ability to cancel an upcoming Appointment or remove a past appointment

4. Appointment Scheduling: Patients can search for available doctors, view their availability, and book appointments at suitable dates and times.

5. Admin Panel: Administrators have administrative privileges to manage the system, including user registration, doctor profiles, patient profiles, appointment management, and generating reports. They can add or remove doctors and patients from the system and oversee the overall functionality.

6. Prescription Management: Doctors can generate and store electronic prescriptions for patients, including details of prescribed medications, dosage instructions, and duration. Patients can access and download their prescriptions for reference and follow-up purposes.

Student Contributions:

Hasaan Ahmad:

Hasaan Ahmad has been instrumental in developing the Doctor class, implementing the login functionality and registration system for doctors, as well as ensuring proper inheritance from the User class. Additionally, Hasaan has played a crucial role in developing the admin panel, designing and implementing its GUI, and adding functionalities for administrators to view and delete doctors, patients, and appointments. His contributions have significantly enhanced the application by providing a seamless user experience and efficient management capabilities within the system.

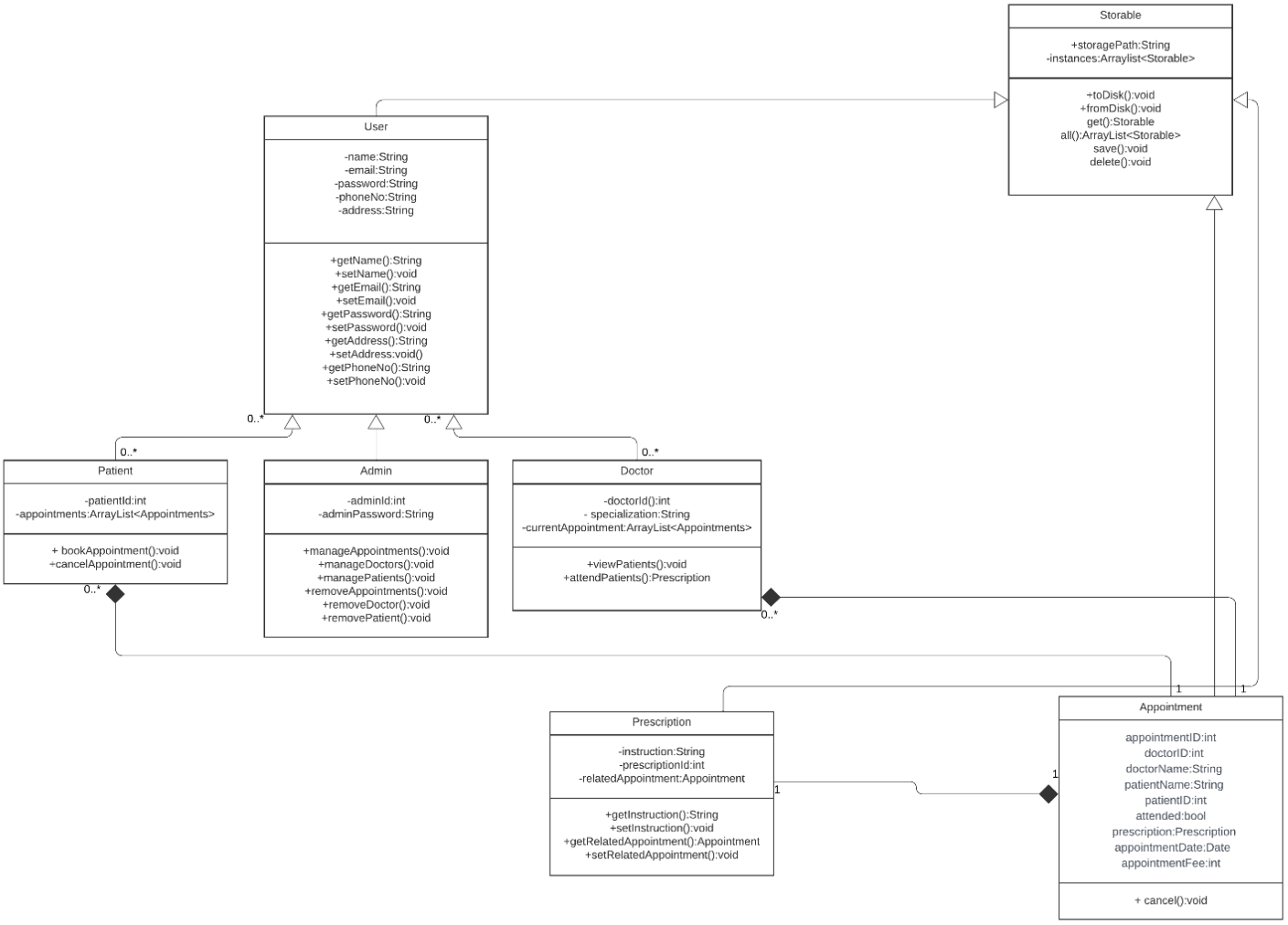
Muhammad Mujtaba:

Muhammad Mujtaba has played a key role in handling file handling operations within the project. He has focused on implementing file input/output operations for storing and retrieving data related to doctors, patients, appointments, and other relevant information. Mujtaba has ensured the proper organization and management of data in files, allowing for efficient data retrieval and storage. Moreover, he has also worked on the prescription handling functionality for doctors, ensuring that prescriptions are properly recorded and associated with the respective patients.

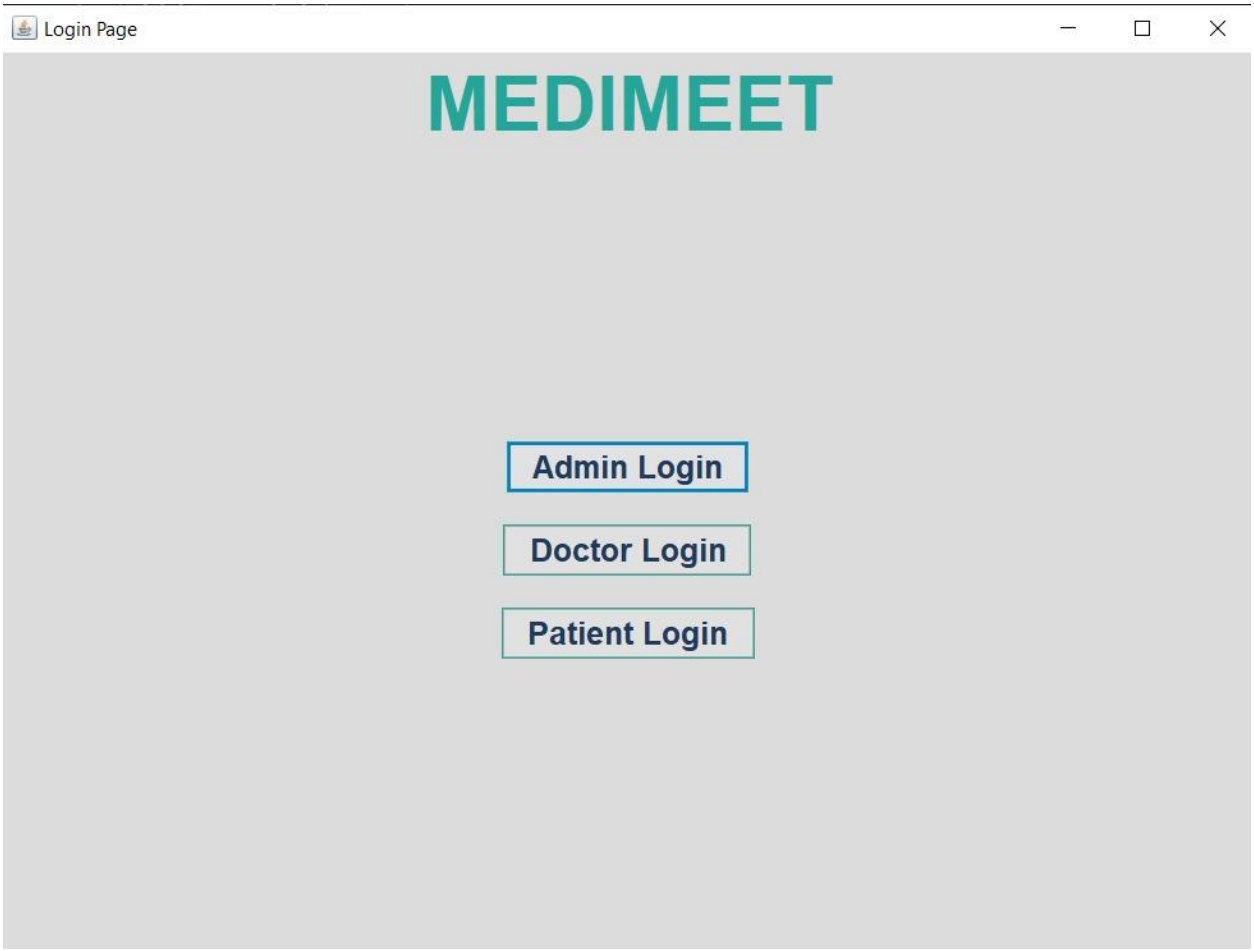
Muhammad Haider Sheikh:

Muhammad Haider Sheikh has taken the lead in developing the graphical user interface (GUI) for the project. He has designed and implemented the user interfaces that provide an intuitive and user-friendly experience for both doctors and patients. Haider has worked on creating interactive screens for viewing appointments, accessing patient information, and managing doctor profiles. Additionally, he has made contributions to the composition aspect of the prescription and appointment classes, ensuring that these classes collaborate effectively to store and retrieve necessary information

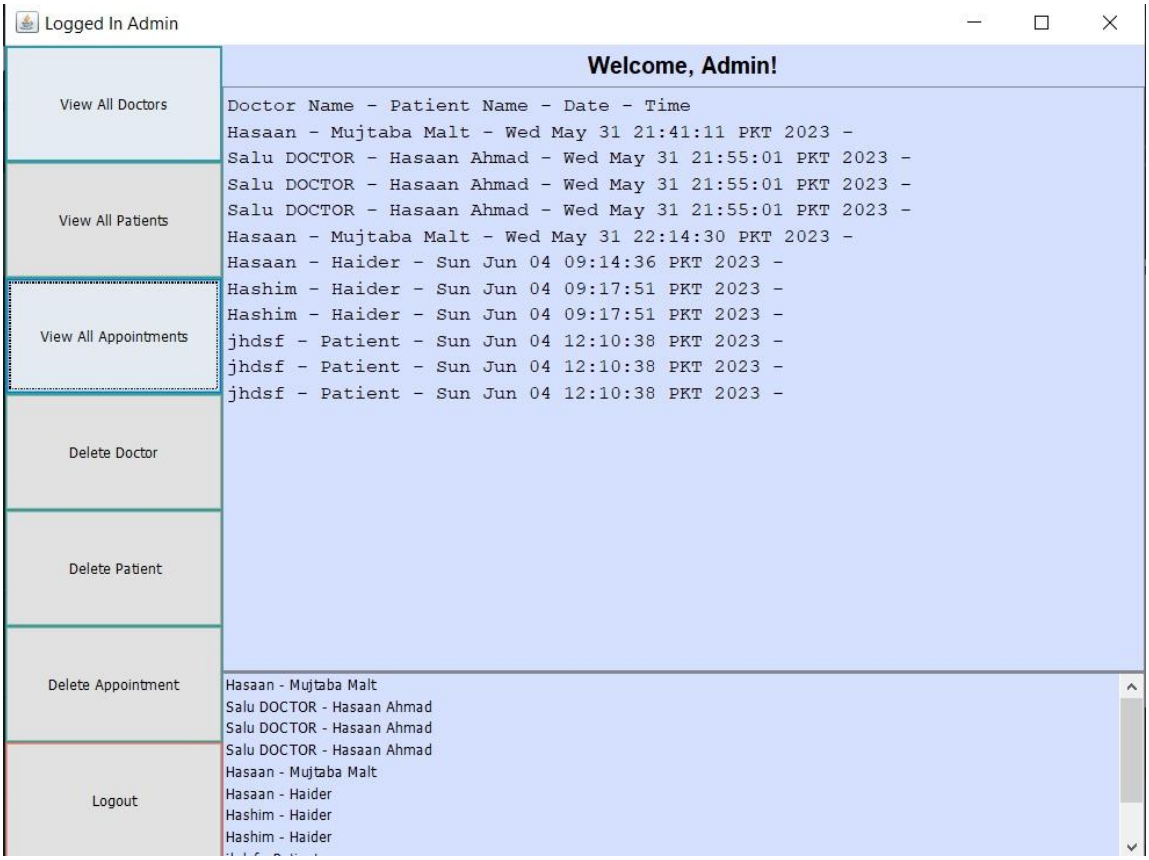
Class Diagram :



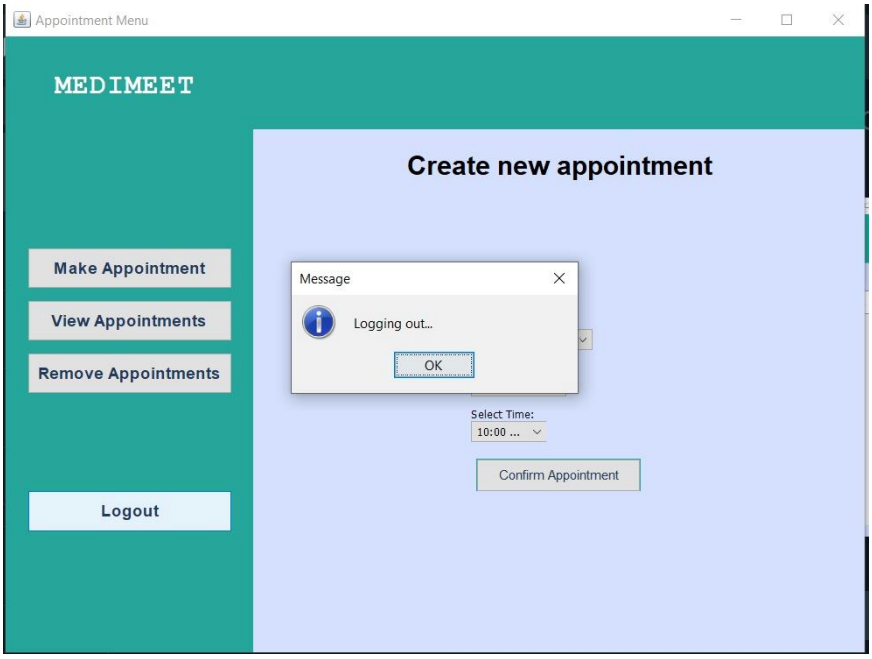
GUI SCREEN SHOTS:
Main Page:



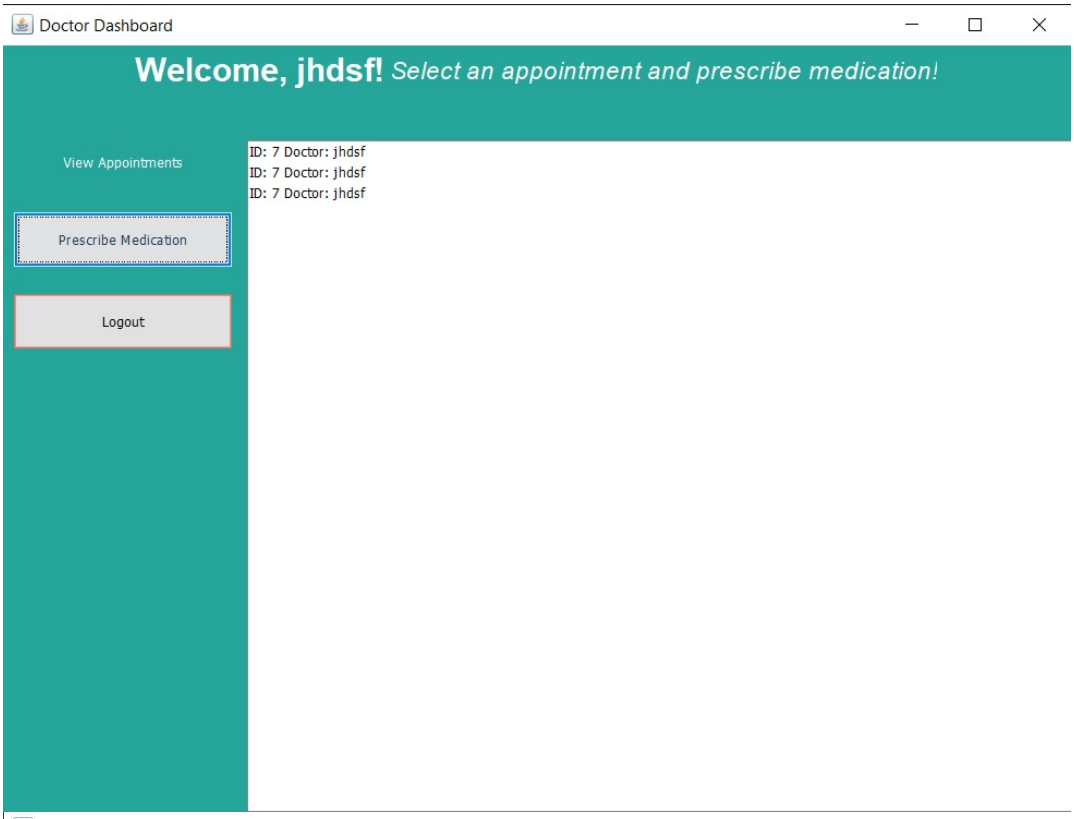
Logged in Admin:

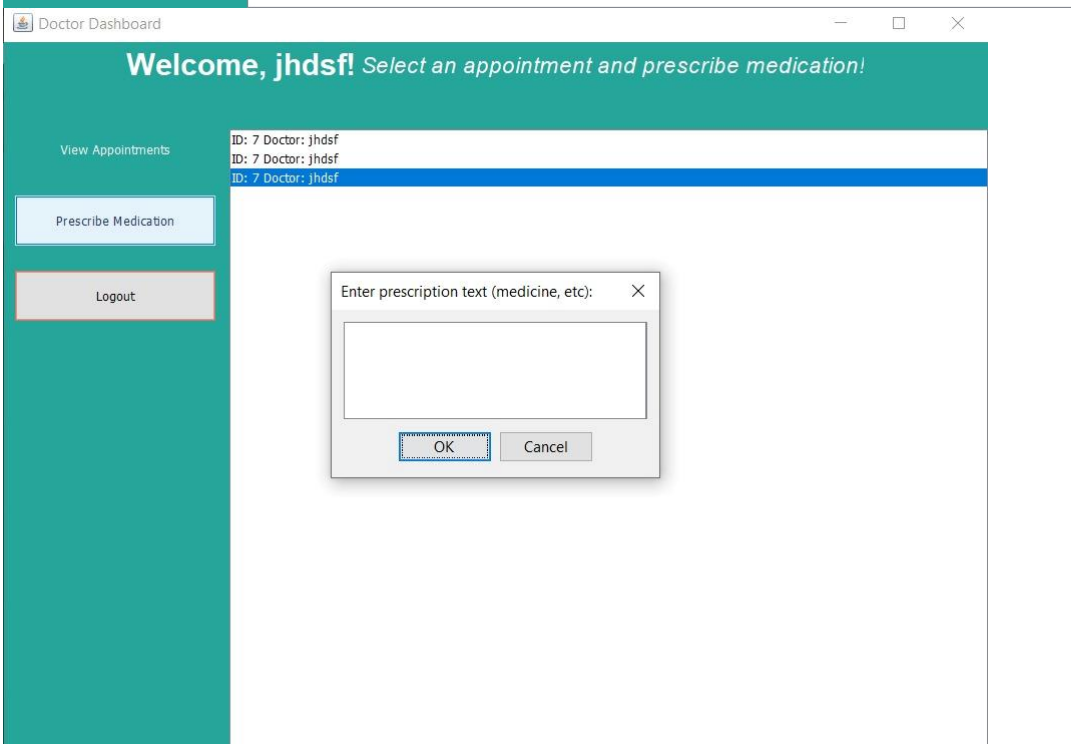


Appointment Menu



Doctor Dashboard and Attending Appointment:





Important Classes:

Storable Class:

```
// ! DO NOT REMOVE COMMENTS

// Author: Mujtaba SP22-BSE-036 muhammadmujtaba150@gmail.com
// Date: May 26, 2023

// Description: Storable module API for the MediMeet application.

// Tests:
// 1. Create Test instance and save to file : PASSED
// 2. Display Test saved instances from file : PASSED
// 3. Separate filter for each sub-class of Storable: PASSED
// 4. Successful deletion of instances: NOT ATTEMPTED
// 5. Filter based on multiple attributes: NOT ATTEMPTED

// Notes: Currently Storable does not include filters other than all().
// However, they will be added sooner.

// ! DO NOT REMOVE COMMENTS

import java.util.*;
import java.io.*;

class Storable implements Serializable {
    public static final long serialVersionUID = 1L;
    public static final String storagePath = "storage.ser";

    private static ArrayList<Storable> instances = new ArrayList<Storable>();

    public static void toDisk() {
        try {

            FileOutputStream fileOutputStream = new
FileOutputStream(storagePath);

            ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
            objectOutputStream.writeObject(instances);

            objectOutputStream.close();
            fileOutputStream.close();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public static void fromDisk() {
        try {
            FileInputStream fileInputStream = new FileInputStream(storagePath);
            ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);
```

```
            instances = (ArrayList<Storable>) objectInputStream.readObject();
            objectInputStream.close();
```

```
            fileInputStream.close();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
```

```
    public static ArrayList<Storable> all(Class<? extends Storable> type) {
        ArrayList<Storable> instancesOfType = new ArrayList<Storable>();
```

```
        for (Storable instance : instances) {
            if (type.isInstance(instance)) {
                instancesOfType.add(instance);
            }
        }
        return instancesOfType;
    }
```

```
    // Returns object of class with ID: null if not found ID is data member of
    // class.
    public static Storable get(Class<? extends Storable> type, int id) {
        ArrayList<Storable> instancesOfType = all(type);
```

```
        for (Storable instance : instancesOfType) {
            if (instance.getId() == id) {
                return instance;
            }
        }
        return null;
    }
```

```
    private int getId() {
        return 0;
```

```
    }

    public void save() {
        instances.add(this);
        Storable.toDisk();
    }

    public void delete() {
        instances.remove(this);
        Storable.toDisk(); // immediately reflect deleted stuff to file
    }

}
```

User Class:

```
//somehow import Storable class - its part of same package so automatically
imported

// At the start of project, call Storable.fromDisk() on top of main method to
load from disk
// At the end of program, call Storable.toDisk() on bottom of main method to save
to disk
// Not call them in each class, just in the main method() once.

/*
WANT TO GET ALL USERS? : ArrayList<Storable> users = Storable.all(User.class);
WANT TO SAVE A NEWLY CREATED USER? : userInstance.save();
*/

import java.io.*;
import java.util.*;

public class User extends Storable {

  private String name;
  private String email;
  private String password;
  private String phone;
  private String address;

  public User(
    String name,
    String email,
    String password,
    String phone,
```

```
    String address
  ) {
    this.name = name;
    this.email = email;
    this.password = password;
    this.phone = phone;
    this.address = address;
  }
```

```
  // Return user if present, else return null - Store this login user in some
  // global variable, and make it null when user logs out
  public static User login(String email, String password) {
    ArrayList<Storable> users = Storable.all(User.class); // filters all
instances belonging to User calss
    for (Storable user : users) {
      User tmp = (User) user;
      if (tmp.getEmail().equals(email) && tmp.getPassword().equals(password)) {
        return tmp;
      }
    }
    return null;
  }
```

```
  public String getName() {
    return name;
  }
```

```
  public void setName(String name) {
    this.name = name;
  }
```

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

```
  public void setEmail(String email) {
    this.email = email;
  }
```

```
  public String getPassword() {
    return password;
  }
```

```
  public void setPassword(String password) {
    this.password = password;
  }
```

```
  public String getPhone() {
    return phone;
  }
```

```
  public void setPhone(String phone) {
    this.phone = phone;
  }
```

```
  public String getAddress() {
    return address;
  }
```

```
  public void setAddress(String address) {
    this.address = address;
  }
```

```
  @Override
  public String toString() {
    return (
      "User [name=" +
      name +
      ", email=" +
      email +
      ", password=" +
      password +
      ", phone=" +
      phone +
      ", address=" +
      address +
      "]"
    );
  }
}
```

Doctor Class:

```
import java.util.ArrayList;
import java.util.Date;
import java.util.Random;
import java.util.Scanner;

public class Doctor extends User {
```

```
  private static final long serialVersionUID = 6403115235281749848L;
```

```
  private int doctorID;
```

```
  private String specialization;
  private ArrayList<Appointment> currentAppointments; // Open appointments which
this doctor is currently part of
```

```
  public Doctor(
    String name,
    String email,
    String password,
    String phone,
    String address,
    int doctorID,
    String specialization,
    ArrayList<Appointment> currentAppointments
  ) {
    super(name, email, password, phone, address);
    this.doctorID = doctorID;
    this.specialization = specialization;
    this.currentAppointments = currentAppointments;
  }
```

```
  // Register a new doctor
  public static void register throws Exception(
    String name,
    String email,
    String password,
    String phone,
    String address,
    int doctorID,
    String specialization
  ) {
    ArrayList<Storable> doctors = Storable.all(Doctor.class);
    if (validID(doctorID)) {
      System.out.println("Doctor ID already exists");
      return;
    } else {
      Doctor newDoctor = new Doctor(
        name,
        email,
        password,
        phone,
        address,
        doctorID,
        specialization,
        null
      );
```

```
      newDoctor.save();
      System.out.println("Doctor registered successfully");
    }
  }
```

```
  public static boolean validID(int id) {
    // Check if doctorID is unique
    ArrayList<Storable> doctors = Storable.all(Doctor.class);
    for (Storable doctor : doctors) {
      Doctor tmp = (Doctor) doctor;
      if (tmp.getDoctorID() == id) {
        return true;
      }
    }
    return false;
  }
```

```
  public static Doctor login(int doctorID, String password) {
    ArrayList<Storable> doctors = Storable.all(Doctor.class);
    for (Storable doctor : doctors) {
      Doctor tmp = (Doctor) doctor;
      if (tmp.getDoctorID() == doctorID && tmp.getPassword().equals(password)) {
        return tmp;
      }
    }
    return null;
  }
```

```
  // Doctor can view all the appointments he is part of and attend patient and
  // assign him with prescription
  public static Prescription attendPatient(Appointment appointment, String
description) {
    // generateing random id
    Random rand = new Random();
    int id = rand.nextInt(1000);
    // making object of perscription
    Prescription prescription = new Prescription(description, appointment);
    // adding prescription to appointment
    appointment.setPrescription(prescription);
    return prescription;
  }
```

```
  // Getter Setters
  public int getDoctorID() {
    return doctorID;
  }
```

```
  public void setDoctorID(int doctorID) {
    this.doctorID = doctorID;
  }
```

```
  public static Doctor getDoctorByName(String name) {
    ArrayList<Storable> doctors = Storable.all(Doctor.class);
    for (Storable doctor : doctors) {
      Doctor tmp = (Doctor) doctor;
      if (tmp.getName().equals(name)) {
        return tmp;
      }
    }
    return null;
  }
```

```
  public String getSpecialization() {
    return specialization;
  }
```

```
  public void setSpecialization(String specialization) {
    this.specialization = specialization;
  }
```

```
  public ArrayList<Appointment> getCurrentAppointments() {
    return currentAppointments;
  }
```

```
  public void setCurrentAppointments(
    ArrayList<Appointment> currentAppointments
  ) {
    this.currentAppointments = currentAppointments;
  }
```

```
  @Override
  public String toString() {
    return this.getName();
  }
}
```

PATIENT CLASS:

```
import java.util.ArrayList;
import java.util.Date;

public class Patient extends User {
  public static final long serialVersionUID = 6149778175131693507L;
```

```
  private int patientID;
  private ArrayList<Appointment> appointments;
```

```
  public Patient(String name, String email, String password, String phone, String
address, int patientID,
      ArrayList<Appointment> appointments) {
    super(name, email, password, phone, address);
    this.patientID = patientID;
    this.appointments = appointments;
  }
```

```
  // Register a new patient
  public static Patient register(String name, String email, String password,
String phone, String address,
      int patientID, Prescription prescription, int doctorID,
ArrayList<Appointment> appointments) {
    ArrayList<Storable> patients = Storable.all(Patient.class);
    if (validID(patientID)) {
      System.out.println("Patient ID already exists");
      return null;
    } else {
      Patient newPatient = new Patient(name, email, password, phone, address,
patientID,
          appointments);
      newPatient.save();
      System.out.println("Patient registered successfully");
      return newPatient;
    }
  }
```

```
  static boolean validID(int id) {
    // Check if patientID is unique
    ArrayList<Storable> patients = Storable.all(Patient.class);
    for (Storable patient : patients) {
      Patient tmp = (Patient) patient;
      if (tmp.getPatientID() == id) {
        return true;
      }
    }
    return false;
  }
```

```
  public static Patient login(int patientID, String password) {
    ArrayList<Storable> patients = Storable.all(Patient.class);
    for (Storable patient : patients) {
```

```
      Patient tmp = (Patient) patient;
      if (tmp.getPatientID() == patientID && tmp.getPassword().equals(password))
{
        return tmp;
      }
    }
    return null;
  }
  // Patient should be able to book an appointment. Appointment for the patient
  // should be stored in patient's appointment array. and in the respective
  // "currentAppointments" array of the doctor under specific doctor id.
```

```
  public void bookAppointment(Doctor doctor, Date date, int fee) {
    // Create a new appointment
    // random appointment id
    int appointmentID = (int) (Math.random() * 1000000);
    Appointment newAppointment = new Appointment(appointmentID, this.patientID,
doctor.getDoctorID(), date, fee);
    newAppointment.save();
    // Add appointment to patient's appointment array
    this.appointments.add(newAppointment);
    // Add appointment to doctor's currentAppointments array
    doctor.getCurrentAppointments().add(newAppointment);
    // Update doctor's currentAppointments array in database
    doctor.save();
  }
```

```
  // Method should return the patient object with id
  public static Patient getPatient(int id) {
    ArrayList<Storable> patients = Storable.all(Patient.class);
    for (Storable patient : patients) {
      Patient tmp = (Patient) patient;
      if (tmp.getPatientID() == id) {
        return tmp;
      }
    }
    return null;
  }
```

```
  // getters and Setters
  public int getPatientID() {
    return patientID;
  }
```

```
  public static Patient getPatientByName(String patientName) {
    ArrayList<Storable> patients = Storable.all(Patient.class);
```

```
    for (Storable patient : patients) {
      Patient tmp = (Patient) patient;
      if (tmp.getName().equals(patientName)) {
        return tmp;
      }
    }
    return null;
  }
```

```
  public void setPatientID(int patientID) {
    this.patientID = patientID;
  }
```

```
  public ArrayList<Appointment> getAppointments() {
    return appointments;
  }
```

```
  public void setAppointments(ArrayList<Appointment> appointments) {
    this.appointments = appointments;
  }
```

```
}
```

Appointment Class:

```
import java.util.ArrayList;
import java.util.Date;
import javax.print.Doc;

import java.io.*;
import java.util.*;
```

```
public class Appointment extends Storable {
```

```
  private static final long serialVersionUID = 8682333983606994514L;
```

```
  private int appointmentID;
  private int doctorID;
  private String doctorName;
  private String patientName;
  private int patientID;
  private boolean attended;
  private Prescription prescription;
  private Date appointmentDate;
  private int appointmentFee;
```

```
  public Appointment(
```

```
    int appointmentID,
    int doctorID,
    int patientID,
    Date appointmentDate,
    int appointmentFee
  ) {
    this.appointmentID = appointmentID;
    this.doctorID = doctorID;
    this.patientID = patientID;
    this.appointmentDate = appointmentDate;
    this.appointmentFee = appointmentFee;
    this.attended = false;
    this.prescription = null;
  }
```

```
  public Appointment(
    int appointmentID,
    String doctorName,
    String patientName,
    Date appointmentDate,
    int appointmentFee
  ) {
    this.appointmentID = appointmentID;
    this.doctorName = doctorName;
    this.patientName = patientName;
    this.appointmentDate = appointmentDate;
    this.appointmentFee = appointmentFee;
  }
```

```
  // getters and Setters
```

```
  public int getAppointmentID() {
    return appointmentID;
  }
```

```
  public void setAppointmentID(int appointmentID) {
    this.appointmentID = appointmentID;
  }
```

```
  public int getDoctorID() {
    return doctorID;
  }
```

```
  public void setDoctorID(int doctorID) {
    this.doctorID = doctorID;
  }
```

```java
  public int getPatientID() {
    return patientID;
  }

  public void setPatientID(int patientID) {
    this.patientID = patientID;
  }

  public Prescription getPrescription() {
    return prescription;
  }

  public void setPrescription(Prescription prescription) {
    this.prescription = prescription;
  }

  public boolean getAttended() {
    return attended;
  }

  public void setAttended(boolean attended) {
    this.attended = attended;
  }

  public Date getAppointmentDate() {
    return appointmentDate;
  }

  public void setAppointmentDate(Date appointmentDate) {
    this.appointmentDate = appointmentDate;
  }

  public int getAppointmentFee() {
    return appointmentFee;
  }

  public void setAppointmentFee(int appointmentFee) {
    this.appointmentFee = appointmentFee;
  }

  // Static method that takes patientID and returns Patient object of that
patient
  public static Patient getPatient(int patientID) {
    return (Patient) Storable.get(Patient.class, patientID);
  }
```

```java
// Static method that takes doctorID and returns Doctor object of that doctor
public static Doctor getDoctor(int doctorID) {
  return (Doctor) Storable.get(Doctor.class, doctorID);
}
```

```java
@Override
public String toString() {
  return "ID: " + appointmentID + " Doctor: " + doctorName;
}
```

```java
public String Display() {
  return "ID: " + appointmentID + " Doctor: " + doctorName;
}
```

```java
public String getDoctorName() {
  return doctorName;
}
```

```java
public void setDoctorName(String doctorName) {
  this.doctorName = doctorName;
}
```

```java
public String getPatientName() {
  return patientName;
}
```

```java
public void setPatientName(String patientName) {
  this.patientName = patientName;
}
```

```java
public static Appointment getAppointmentByNames(
  String doctorName,
  String patientName
) {
  ArrayList<Storable> appointments = Storable.all(Appointment.class);
  for (Storable appointment : appointments) {
    Appointment tmp = (Appointment) appointment;
    if (
      tmp.getDoctorName().equals(doctorName) &&
      tmp.getPatientName().equals(patientName)
    ) {
      return tmp;
    }
  }
  return null;
}
```

```
  // Print all data related to this appointment to a text file including
perscription,
  // everything in perfect format so that it can be printed and given to the
patient:
```

```
  void printAllInformationToTextFile(){
    String fileName = "Appointment" + this.getAppointmentID() + ".txt";
    String data = "Appointment ID: " + this.getAppointmentID() + "\n" +
                  "Doctor Name: " + this.getDoctorName() + "\n" +
                  "Patient Name: " + this.getPatientName() + "\n" +
                  "Appointment Date: " + this.getAppointmentDate() + "\n" +
                  "Appointment Fee: " + this.getAppointmentFee() + "\n" +
                  "Prescription: " + this.getPrescription().getInstruction() +
"\n";
    try {
      FileWriter myWriter = new FileWriter(fileName);
      myWriter.write(data);
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred while eriting appointment and
perscription to file.");
      e.printStackTrace();
    }
  }
```

```
}
```

Prescription Class:

```
import java.sql.Date;

public class Prescription extends Storable {
  private static final long serialVersionUID = -7756106466251150463L;
```

```
  private String instruction;
  private int prescriptionId;
  private Appointment relatedAppointment;
```

```
  public Prescription(String instruction, Appointment relatedAppointment) {
    this.instruction = instruction;
    this.prescriptionId = (int) (Math.random() * 1000000) +
        (int) (Math.random() * 1000) -
```

```
        (int) (Math.random() * 1000);
    this.relatedAppointment = relatedAppointment;
}
```

```
public Prescription() {
    this.instruction = "";
    this.prescriptionId = (int) (Math.random() * 1000000) +
        (int) (Math.random() * 1000) -
        (int) (Math.random() * 1000);
    this.relatedAppointment = null;
}
```

```
public String getInstruction() {
    return instruction;
}
```

```
public void setInstruction(String instruction) {
    this.instruction = instruction;
}
```

```
public int getPrescriptionId() {
    return prescriptionId;
}
```

```
public Appointment getRelatedAppointment() {
    return relatedAppointment;
}
```

```
public void setRelatedAppointment(Appointment relatedAppointment) {
    this.relatedAppointment = relatedAppointment;
}
```

```
@Override
public String toString() {
    return ("Prescription ID: " +
        prescriptionId +
        "\n" +
        "Instruction: " +
        instruction +
        "\n" +
        "Related Appointment: " +
        relatedAppointment +
        "\n" +
        "By doctor: " +
        relatedAppointment.getDoctorName() +
        "To patient: " +
```

```
        relatedAppointment.getPatientName() +
        "Dated: " +
        new Date(System.currentTimeMillis()).toString() +
        "\n");
  }
}
```