```
import java.text.DecimalFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.temporal.ChronoUnit;
import java.util.Calendar;
import java.util.Date;
import java.util.concurrent.TimeUnit;

class Accountant {


    double charge = 0;

    /**
     * Generates the charge for the specified patient for staying in the given room
     * based on the number of days stayed.
     *
     * @param patient the patient in the room
     * @param room the room that the patient has stayed in
     * @return the resulting charge for the patient
     */

    public String generateRoomCharge(Patient patient, Room room) {
        Calendar cal = Calendar.getInstance();
        String day = cal.get(Calendar.DAY_OF_MONTH) + "";

        String month;


        if (cal.get(Calendar.MONTH) + 1 > 9)
            month = "" + (cal.get(Calendar.MONTH) + 1);
        else
            month = "0" + (cal.get(Calendar.MONTH) + 1);

        String year = "" + cal.get(Calendar.YEAR);

        int date = Integer.parseInt(day + month + year);

        SimpleDateFormat myFormat = new SimpleDateFormat("ddMMyyyy");

        String inputString1;
        if((date+"").length() == 7)
            inputString1 = "0" + date+"";
        else
            inputString1 = date+"";


        String inputString2;
        if((patient.getDateOfRoomAdmittance()+"").length() == 7)
            inputString2 = "0" + patient.getDateOfRoomAdmittance()+"";
        else
            inputString2 = patient.getDateOfRoomAdmittance()+"";

        int days=0;
        try {
            Date date1 = myFormat.parse(inputString1);
            Date date2 = myFormat.parse(inputString2);
            days = (int)ChronoUnit.DAYS.between(date2.toInstant(),date1.toInstant());
        } catch (ParseException e) {
            e.printStackTrace();
        }

        charge = room.getCostPerDay()*(days+1);

        return new DecimalFormat("##.00").format(charge);
    }
```

```
}
```

```
public class Appointment extends Task{

    String appointmentType;
    Patient patient;

    /**
     * Constructs a new Appointment instance using the specified information.
     *
     * @param purpose the purpose of the appointment
     * @param description the description of what will occur during the appointment
     * @param appointmentType the type of appointment
     * @param doctor the doctor involved in the appointment
     * @param patient the patient involved in the appointment
     * @param date the date of the appointment
     * @param startTime the starting time of the appointment
     * @param endTime the estimated ending time of the appointment
     */
    public Appointment(String purpose, String description, String appointmentType, Doctor doctor,
Patient patient,
                       int date, int startTime, int endTime) {
        super(purpose, description, doctor, date, startTime, endTime);
        this.appointmentType = appointmentType;
        this.patient = patient;
    }

    /**
     * Returns the patient involved in this appointment.
     *
     * @return the appointment involved in this appointment
     */
    public Patient getPatient() {
        return patient;
    }

    /**
     * Returns the type of this appointment.
     *
     * @return the type of this appointment
     */
    public String getAppointmentType() {
        return appointmentType;
    }

}
```

```
import java.util.ArrayList;
import java.util.UUID;

public class Doctor {


    private String firstName;
    private String lastName;
```

```
    private String email;
    private long number;
    private String uniqueID;
    private ArrayList<Task> tasks;
    private String address;

    /**
     * Constructs a new Doctor using the specified information.
     *
     * @param firstName the first name of the doctor
     * @param lastName  the last name of the doctor
     * @param email     the email of the doctor
     * @param number    the telephone number of the doctor
     */
    public Doctor(String firstName, String lastName, String email, long number, String address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.number = number;
        uniqueID = UUID.randomUUID().toString();
        tasks = new ArrayList<>();
        this.address = address;
    }

    /**
     * Constructs a new Doctor using the specified information.
     *
     * @param firstName the first name of the doctor
     * @param lastName  the last name of the doctor
     * @param email     the email of the doctor
     * @param number    the telephone number of the doctor
     * @param uniqueID  the unique ID of the doctor
     */
    public Doctor(String firstName, String lastName, String email, long number, String address,
String uniqueID) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.number = number;
        this.address= address;
        this.uniqueID = uniqueID;
        tasks = new ArrayList<>();
    }

    /**
     * Returns the address of this doctor.
     *
     * @return the adress of this doctor
     */
    public String getAddress() {
        return address;
    }

    /**
     * Sets the address of this doctor.
     *
     * @param address the new address of this doctor
     */
    public void setAddress(String address) {
        this.address = address;
    }

    /**
     * Returns the first name of this doctor.
     *
     * @return the first name of this doctor
     */
    public String getFirstName() {
        return firstName;
```

```
}

/**
 * Sets the first name of this doctor.
 *
 * @param firstName the new first name of this doctor
 */
public void setFirstName(String firstName) {
    this.firstName = firstName;
}

/**
 * Returns the last name of this doctor.
 *
 * @return the last name of this doctor
 */
public String getLastName() {
    return lastName;
}

/**
 * Sets the last name of this doctor.
 *
 * @param lastName the new last name of this doctor
 */
public void setLastName(String lastName) {
    this.lastName = lastName;
}

/**
 * Returns the email of this doctor.
 *
 * @return the email of this doctor
 */
public String getEmail() {
    return email;
}

/**
 * Sets the email of this doctor.
 *
 * @param email the new email of this doctor
 */
public void setEmail(String email) {
    this.email = email;
}

/**
 * Returns the telephone number of this doctor.
 *
 * @return the telephone number of this doctor
 */
public long getNumber() {
    return number;
}

/**
 * Sets the telephone number of this doctor.
 *
 * @param number the new telephone number of this doctor
 */
public void setNumber(long number) {
    this.number = number;
}

/**
 * Returns the unique ID of this doctor.
 *
 * @return the unique ID of this doctor
```

```
     */
    public String getUniqueID() {
        return uniqueID;
    }

    /**
     * Returns the tasks of this doctor.
     *
     * @return the tasks of this doctor
     */
    public ArrayList<Task> getTasks() {
        return tasks;
    }
}




public class Examination {


   private int date;
   private String symptom;
   private String diagnosis;
   private String treatment;
   private String remarks;
   private Examination next;

   /**
    * Creates a new examination with the specified information.
    *
    * @param date the date that this examination was performed
    * @param symptom the symptoms discovered during this examination
    * @param diagnosis the information about the diagnosis performed during this examination
    * @param treatment the information about the treatment discussed during this examination
    * @param remarks any remarks regarding this examination
    */
   public Examination(int date, String symptom, String diagnosis, String treatment, String
remarks) {
      this.date = date;
      this.symptom = symptom;
      this.diagnosis = diagnosis;
      this.treatment = treatment;
      this.remarks = remarks;
   }


   /**
    * Returns the date that this examination was performed.
    *
    * @return the date that this examination was performed
    */
   public int getDate() {
      return date;
   }

   /**
    * Returns the symptoms discovered during this examination.
    *
    * @return the symptoms discovered during this examination
    */
   public String getSymptom() {
      return symptom;
   }

   /**
    * Returns the information about the diagnosis performed during this examination.
    *
```

```
     * @return the information about the diagnosis performed during this examination
     */
    public String getDiagnosis() {
        return diagnosis;
    }

    /**
     * Returns the information about the treatment discussed during this examination.
     * @return the information about the treatment discussed during this examination
     */
    public String getTreatment() {
        return treatment;
    }

    /**
     * Returns any remarks regarding this examination.
     *
     * @return any remarks regarding this examination
     */
    public String getRemarks() {
        return remarks;
    }


    /**
     * Return the next examination performed after this examination.
     *
     * @return the next examination performed after this examination
     */
    public Examination getNext()
    {
        return next;
    }

    /**
     * Adds the new examination performed after this examination.
     *
     * @param examination the examination performed after this examination
     */
    public void setNext(Examination examination)
    {
        next = examination;
    }


}
```

```
import jdk.nashorn.internal.scripts.JO;
import sun.awt.image.codec.JPEGParam;

import javax.swing.*;
import javax.swing.border.Border;
import javax.swing.border.CompoundBorder;
import javax.swing.border.EmptyBorder;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.lang.reflect.Array;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
```

```java
import java.util.*;

/**
 * Created by kshitij314 on 2017-12-24.
 */
class GUI {
    private static final int ADD_APPOINTMENT_FRAME_HEIGHT = 160;
    private static final int ADD_APPOINTMENT_FRAME_WIDTH = 630;
    private static final int ADD_DOCTOR_FRAME_HEIGHT = 198;
    private static final int ADD_DOCTOR_FRAME_WIDTH = 600;
    private static final int ADD_LAB_TEST_FRAME_HEIGHT = 200;
    private static final int ADD_LAB_TEST_FRAME_WIDTH = 510;
    private static final int ADD_MEDICINE_FRAME_HEIGHT = 245;
    private static final int ADD_MEDICINE_FRAME_WIDTH = 400;
    private static final int ADD_PATIENT_FRAME_HEIGHT = 270;
    private static final int ADD_PATIENT_FRAME_WIDTH = 500;
    private static final int ADD_PATIENT_OWNER_FRAME_HEIGHT = 230;
    private static final int ADD_PATIENT_OWNER_FRAME_WIDTH = 500;
    private static final int ADD_PATIENT_ROOM_FRAME_HEIGHT = 100;
    private static final int ADD_PATIENT_ROOM_FRAME_WIDTH = 280;
    private static final int DISPLAY_HISTORY_FRAME_HEIGHT = 240;
    private static final int DISPLAY_HISTORY_FRAME_WIDTH = 510;
    private static final int EDIT_DOCTOR_FRAME_HEIGHT = 300;
    private static final int EDIT_DOCTOR_FRAME_WIDTH = 600;
    private static final int EDIT_DOCTOR_GENERAL_FRAME_HEIGHT = 170;
    private static final int EDIT_DOCTOR_GENERAL_FRAME_WIDTH = 600;
    private static final int EDIT_MEDICINE_FRAME_HEIGHT = 300;
    private static final int EDIT_MEDICINE_FRAME_WIDTH = 400;
    private static final int EDIT_OWNER_INFO_FRAME_HEIGHT = 185;
    private static final int EDIT_OWNER_INFO_FRAME_WIDTH = 600;
    private static final int EDIT_PATIENT_FRAME_HEIGHT = 500;
    private static final int EDIT_PATIENT_FRAME_WIDTH = 1100;
    private static final int EDIT_PATIENT_GENERAL_FRAME_HEIGHT = 210;
    private static final int EDIT_PATIENT_GENERAL_FRAME_WIDTH = 490;
    private static final int EDIT_ROOMS_FRAME_HEIGHT = 280;
    private static final int EDIT_ROOMS_FRAME_WIDTH = 400;
    private static final int MAIN_FRAME_HEIGHT = 480;
    private static final int MAIN_FRAME_WIDTH = 720;
    private static final int SORT_BY_DATE = 0;
    private static final int POSSIBLE_LAB_TESTS_HEIGHT = 220;
    private static final int POSSIBLE_LAB_TESTS_WIDTH = 480;
    private static final int SORT_BY_DOCTORS = 2;
    private static final int SORT_BY_PATIENTS = 1;

    private static final String ADD = "Add";
    private static final String ADDRESS = "Address:";
    private static final String ADD_DOCTOR = "Add Doctor";
    private static final String ADD_MEDICINE = "Add Medicine";
    private static final String ADD_PATIENT = "Add Patient";
    private static final String APPOINTMENT = "Appointment:";
    private static final String APPOINTMENT_DESCRIPTION = "Appointment Description:";
    private static final String APPOINTMENT_INFORMATION = "Appointment Information";
    private static final String APPOINTMENT_TYPE = "Appointment Type:";
    private static final String ASSIGN_ROOM = "Assign Room";
    private static final String ASSIGNED_ROOM = "Assigned Room:";
    private static final String ASSIGN_DOCTOR_AUTOMATICALLY = "Assign Doctor Automatically";
    private static final String ATTACH_REPORT_IF_AVAILABLE = "*Attach Report if Available";
    private static final String BIRTHDATE = "Birthdate:";
    private static final String BREED = "Breed:";
    private static final String CANCEL = "Cancel";
    private static final String REMOVE_APPOINTMENT = "Remove Appointment";
    private static final String CHANGE_STATUS = "Change Status";
    private static final String CHOOSE_FILE = "Choose File";
    private static final String COLOUR = "Colour:";
    private static final String COST = "Cost";
    private static final String COST_PER_DAY = "Cost per Day:";
    private static final String CREATE_NEW_APPOINTMENT = "Create New Appointment";
    private static final String CREATE_TASK = "Create Task";
    private static final String DATE = "Date: ";
```

```
    private static final String DATE_FORMAT_DASH = "dd/mm/yyyy";
    private static final String DATE_FORMAT_PLAIN = "ddmmyyyy";
    private static final String DAY = "Day:";
    private static final String DESCRIPTION = "Description:";
    private static final String DIAGNOSIS = "Diagnosis:";
    private static final String DOCTOR = "Doctor";
    private static final String DOCTOR_INFORMATION = "Doctor Information";
    private static final String EDIT = "Edit";
    private static final String EDIT_LABORATORY_TESTS = "Edit Laboratory Tests";
    private static final String EDIT_ROOMS = "Edit Rooms";
    private static final String EMAIL = "Email: ";
    private static final String EMPTY_TEXTFIELD_MESSAGE = "Please ensure that all the fields are
filled.";
    private static final String END_TIME = "End Time:";
    private static final String ENTER = "Enter";
    private static final String ERROR = "Error";
    private static final String EXAMINATION_INFORMATION = "Examination Information";
    private static final String EXIT = "Exit";
    private static final String FINISH = "Finish";
    private static final String FIRST_NAME = "First Name:";
    private static final String FIRST_TAB_TOP_DISPLAY = "Doctors & Patients";
    private static final String FOURTH_TAB_TOP_DISPLAY = "Settings";
    private static final String FORMAT_BIRTHDATE_CORRECT = "Please format the birthdate
correctly: dd/mm/yyyy";
    private static final String GENERAL_INFORMATION = "General Information";
    private static final String GENERATE_BILL = "Discharge Patient & Generate Bill";
    private static final String HISTORY = "History";
    private static final String INCORRECT_TEXTFIELD_MESSAGE = "Please ensure that all the
information provided is appropriately formatted.";
    private static final String LABORATORY_TEST_INFORMATION = "Laboratory Test Information";
    private static final String LABORATORY_TESTS = "Laboratory Tests";
    private static final String LABORATORY_TEST_NAME = "Laboratory Test Name:";
    private static final String LAST_NAME = "Last Name:";
    private static final String MAIN_PURPOSE = "Main Purpose:";
    private static final String MAX_PAST_EXAMINATIONS = "Maximum Number of Past Examinations for
a Patient:";
    private static final String MEDICINE_INFORMATION = "Medicine Information";
    private static final String MONTH = "Month:";
    private static final String NAME = "Name:";
    private static final String NEW_APPOINTMENT = "New Appointment";
    private static final String NEXT = "Next";
    private static final String NO_DOCTORS_AVAILABLE = "No doctors are available during the
specified date and time.";
    private static final String NO_ROOMS_AVAILIABLE = "No rooms availiable.";
    private static final String NOT_ENOUGH_MEDICINE = "Not enough medicine in stock.";
    private static final String OWNER = "Owner: ";
    private static final String OWNER_INFORMATION = "Owner Information";
    private static final String OWNER_NOT_IN_SYSTEM = "*Leave blank if owner is not yet in the
system";
    private static final String PATIENT = "Patient";
    private static final String PATIENT_INFORMATION = "Patient Information";
    private static final String QUANTITY = "Quantity:";
    private static final String REMARKS = "Remarks";
    private static final String REMOVE = "Remove";
    private static final String REMOVE_PATIENTS_IN_ROOM = "Please remove the patients in this
room first.";
    private static final String ROOM = "Room:";
    private static final String ROOM_INFORMATION = "Room Information";
    private static final String ROOM_NAME = "Room Name:";
    private static final String SAVE = "Save";
    private static final String SEARCH = "Search:";
    private static final String SECOND_TAB_TOP_DISPLAY = "Appointments";
    private static final String SELECT_APPOINTMENT_FIRST = "Please select an appointment first.";
    private static final String SELECT_EXAMINATION_FIRST = "Please select an examination first.";
    private static final String SELECT_LABORATORY_TEST_FIRST = "Please select a laboratory test
first.";
    private static final String SELECT_ROOM_FIRST = "Please select a room first.";
    private static final String SELECT_TASK_FIRST = "Please select a task first.";
    private static final String SEX = "Sex:";
```

```
    private static final String START_TIME = "Start time:";
    private static final String STATUS = "Status:";
    private static final String STATUS_CANNOT_BE_CHANGED = "The laboratory test is already
completed. The status cannot be further changed.";
    private static final String SORT_BY = "Sort by:";
    private static final String SPECIES = "Species: ";
    private static final String SPOTS_AVAILABLE = "Spots Available:";
    private static final String SYMPTOMS = "Symptoms:";
    private static final String TASKS = "Tasks";
    private static final String TASK_DESCRIPTION = "Task Description:";
    private static final String TASK_INFORMATION = "Task Information";
    private static final String TELEPHONE_NUMBER = "Telephone Number:";
    private static final String TEST_TYPE = "Test Type:";
    private static final String TIME = "Time: ";
    private static final String THIRD_TAB_TOP_DISPLAY = "Inventory";
    private static final String TOTAL_COST_FOR_ROOM = "The total cost for the room is ";
    private static final String TREATMENT = "Treatment:";
    private static final String UPCOMING_APPOINTMENTS = "Upcoming Appointments";
    private static final String VIEW = "View";
    private static final String WEIGHT = "Weight(lb):";
    private static final String YEAR = "Year:";
    private static final String[] appointmentColumns = {"Purpose", "Date", "Time"};
    private static final String[] dayOrNight = {"AM", "PM"};
    private static final String[] historyColumns = {"Date", "Symptom", "Diagnosis", "Treatment",
"Remarks"};
    private static final String[] labColumns = {"Test Type", "Date", "Status"};
    private static final String[] months = {"January", "February", "March", "April", "May",
"June", "July", "August", "September",
            "October", "November", "December"};
    private static final String[] minutes = {"00", "15", "30", "45"};
    public static final String ASSIGN_ROOM_AUTOMATICALLY = "Assign Room Automatically";
    public static final String NONE = "none";
    public static final String[] statusList = {"Incompleted","In progress","Completed"};

    public static int maximumExaminationsDisplayed = 5;
    public Manager manager = new Manager();
    private JFrame mainFrame;
    private JFrame addAppointmentFrame;
    private JFrame addDoctorFrame;
    private JFrame addTaskFrame;
    private JFrame addPatientExaminationFrame;
    private JFrame addPatientFrame;
    private JFrame addPatientLabTestFrame;
    private JFrame addPatientRoomFrame;
    private JFrame addStockFrame;
    private JFrame editPatientFrame;
    private JFrame editDoctorFrame;
    private JFrame editMedicineFrame;
    private JFrame editRoomsFrame;
    private JFrame editPossibleLabTestsFrame;
    private JFrame editDoctorGeneralInfoFrame;
    private JFrame editPatientGeneralInfoFrame;
    private JFrame editOwnerInfoFrame;
    private JFrame showAppointmentInfoFrame;
    private JFrame displayHistoryFrame;
    private JPanel patientBoxes = new JPanel();
    private ArrayList<JButton> patientBoxPanels = new ArrayList<JButton>();
    private JPanel doctorBoxes = new JPanel();
    private ArrayList<JButton> doctorBoxPanels = new ArrayList<JButton>();
    private JPanel medicineBoxes = new JPanel();
    private ArrayList<JButton> medicineBoxPanels = new ArrayList<JButton>();
    private JPanel appointmentBoxes = new JPanel();
    private ArrayList<JButton> appointmentBoxPanels = new ArrayList<JButton>();
    private JPanel firstTabContents = new JPanel();
    private JPanel secondTabContents = new JPanel();
    private JPanel thirdTabContents = new JPanel();
    private JPanel fourthTabContents = new JPanel();
    private JTabbedPane topTabs;
    private JButton searchPatient;
```

```
private JScrollPane pane;
private JButton searchDoctor;
private String[] testTypes;
private JButton doctorButton;
private JButton enterMedicineButton;

public static void main(String args[]) {

    try {
        for (UIManager.LookAndFeelInfo info : UIManager.getInstalledLookAndFeels()) {
            if ("Nimbus".equals(info.getName())) {
                UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (Exception e) {

    }
    GUI gui = new GUI();

    // Obtains the previously saved data from text files.
    gui.getInitialData();
    // Creates the graphical user interface.
    gui.createMenu();


}

/**
 * Loads the information previously stored in text files.
 */
private void getInitialData() {
    try {
        manager.loadFileData();
    } catch (IOException e) {
        e.printStackTrace();
    }
    firstTabContents.setLayout(new BorderLayout());
    secondTabContents.setLayout(new BorderLayout());
    thirdTabContents.setLayout(new BorderLayout());
    fourthTabContents.setLayout(new BorderLayout());
}

/**
 * Loads the frame used to display the main menu.
 */
private void createMenu() {
    mainFrame = new JFrame();
    mainFrame.setLayout(new BorderLayout());
    createTopTabs();
    createTopOptionsPatDoc(null);

    mainFrame.setSize(new Dimension(GUI.MAIN_FRAME_WIDTH, GUI.MAIN_FRAME_HEIGHT));
    mainFrame.setLocationRelativeTo(null);
    mainFrame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    searchPatient.doClick();
    mainFrame.setVisible(true);
    WindowListener exitListener = new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            manager.storeFileData();
            System.exit(0);
        }
    };
    mainFrame.setResizable(false);
    mainFrame.addWindowListener(exitListener);
}

/**
```

```java
 * Creates the four tabs on top of the frame.
 */
private void createTopTabs() {
    topTabs = new JTabbedPane();
    mainFrame.getContentPane().add(topTabs, BorderLayout.NORTH);

    // Adds all the tabs and their respective contents to a tabbed pane.
    topTabs.add(FIRST_TAB_TOP_DISPLAY, firstTabContents);
    topTabs.add(SECOND_TAB_TOP_DISPLAY, secondTabContents);
    topTabs.add(THIRD_TAB_TOP_DISPLAY, thirdTabContents);
    topTabs.add(FOURTH_TAB_TOP_DISPLAY, fourthTabContents);
    topTabs.setTabLayoutPolicy(JTabbedPane.SCROLL_TAB_LAYOUT);

    // Loads the required information based on the tab selected.
    topTabs.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            if (topTabs.getSelectedIndex() == 0) {
                firstTabContents.removeAll();
                removeAllBoxes();
                createTopOptionsPatDoc(null);
                createPatientsDisplayFeed(manager.getPatients());
                firstTabContents.revalidate();
            } else if (topTabs.getSelectedIndex() == 1) {
                removeAllBoxes();
                createTopOptionsApp();
                createAppointmentsDisplayFeed(SORT_BY_DATE);
            } else if (topTabs.getSelectedIndex() == 2) {
                removeAllBoxes();
                createTopOptionsMedicine(null);
                createStockManagementsDisplayFeed(manager.getPharmacist().getMedicines());
            } else {
                removeAllBoxes();
                createSettingDisplay();
            }
        }
    });
}

/**
 * Removes all the display feeds from each tab.
 */
private void removeAllBoxes() {
    patientBoxes = new JPanel();
    patientBoxPanels = new ArrayList<JButton>();
    doctorBoxes = new JPanel();
    doctorBoxPanels = new ArrayList<JButton>();
    appointmentBoxes = new JPanel();
    appointmentBoxPanels = new ArrayList<JButton>();
    medicineBoxes = new JPanel();
    medicineBoxPanels = new ArrayList<JButton>();
}

/**
 * Creates the options allowing the user to search and add patients and doctor.
 * This is only displayed if the first tab is selected.
 *
 * @param searchedText the input text used to search for a patient/doctor
 */
private void createTopOptionsPatDoc(String searchedText) {
    JPanel bottomOptions = new JPanel();
    firstTabContents.add(bottomOptions, BorderLayout.PAGE_START);

    JPanel search = new JPanel();
    bottomOptions.add(search);

    JLabel searchLabel = new JLabel(SEARCH);
    search.add(searchLabel);

    JTextField searchField = new JTextField(8);
```

```
        search.add(searchField);

        if (searchedText != null)
            searchField.setText(searchedText);

        // Adds the button used to search for the doctor.
        searchDoctor = new JButton(DOCTOR);
        search.add(searchDoctor);
        searchDoctor.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ArrayList<Doctor> searchedDoctors = manager.searchDoctor(searchField.getText());
                patientBoxes = new JPanel();
                patientBoxPanels = new ArrayList<JButton>();
                doctorBoxes = new JPanel();
                doctorBoxPanels = new ArrayList<JButton>();
                firstTabContents.removeAll();
                createTopOptionsPatDoc(searchField.getText());
                createDoctorsDisplayFeed(searchedDoctors);
                firstTabContents.revalidate();
            }
        });

        // Adds the button used to search for the patient.
        searchPatient = new JButton(PATIENT);
        search.add(searchPatient);
        searchPatient.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                ArrayList<Patient> searchedPatients =
manager.searchPatientWithRooms(searchField.getText());
                patientBoxes = new JPanel();
                patientBoxPanels = new ArrayList<JButton>();
                doctorBoxes = new JPanel();
                doctorBoxPanels = new ArrayList<JButton>();
                firstTabContents.removeAll();
                createTopOptionsPatDoc(searchField.getText());
                createPatientsDisplayFeed(searchedPatients);
                firstTabContents.revalidate();
            }
        });

        JPanel addDocPat = new JPanel();
        bottomOptions.add(addDocPat);

        // Adds the button used to add another doctor in the program.
        JButton addDoctor = new JButton(ADD_DOCTOR);
        addDocPat.add(addDoctor);
        addDoctor.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createAddDoctorDisplay();
            }
        });

        // Adds the button used to add another patient in the program.
        JButton addPatient = new JButton(ADD_PATIENT);
        addDocPat.add(addPatient);
        addPatient.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createAddPatientDisplay();
            }
        });
    }

    /**
     * Creates the frame displayed when adding a new doctor.
     */
    private void createAddDoctorDisplay() {
        addDoctorFrame = new JFrame();
        addDoctorFrame.setLayout(new BorderLayout());
        addDoctorFrame.setAlwaysOnTop(true);
```

```
        mainFrame.setEnabled(false);
        mainFrame.setAlwaysOnTop(false);
        createAddDoctorInformationForm();

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addDoctorFrame.dispose();
                addDoctorFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };

        addDoctorFrame.addWindowListener(exitListener);
        addDoctorFrame.setSize(new Dimension(GUI.ADD_DOCTOR_FRAME_WIDTH,
GUI.ADD_DOCTOR_FRAME_HEIGHT));
        addDoctorFrame.setLocationRelativeTo(null);
        addDoctorFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display shown when adding a new doctor which allows
information
     * to be entered about the new doctor.
     */
    private void createAddDoctorInformationForm() {
        JPanel informationFields = new JPanel();
        addDoctorFrame.getContentPane().add(informationFields, BorderLayout.CENTER);

        JPanel doctorInformation = new JPanel();
        doctorInformation.setLayout(new FlowLayout());
        addDoctorFrame.getContentPane().add(doctorInformation, BorderLayout.NORTH);

        JLabel doctorInformationLabel = new JLabel(DOCTOR_INFORMATION);
        doctorInformation.add(doctorInformationLabel);

        Border border = informationFields.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        informationFields.setBorder(new CompoundBorder(border, margin));

        informationFields.setLayout(new GridLayout(0, 4));

        String[] labels = {FIRST_NAME, LAST_NAME, EMAIL, TELEPHONE_NUMBER, ADDRESS};
        ArrayList<JTextField> textFields = new ArrayList<JTextField>();

        // Creates the form for the user to enter information using textfields.
        for (int i = 0; i < labels.length; i++) {
            informationFields.add(new JLabel(labels[i]));
            textFields.add(new JTextField(15));
            informationFields.add(textFields.get(i));
        }

        JPanel bottomButtons = new JPanel();
        addDoctorFrame.getContentPane().add(bottomButtons, BorderLayout.SOUTH);
        bottomButtons.setLayout(new BorderLayout());

        JButton cancelButton = new JButton(CANCEL);
        bottomButtons.add(cancelButton, BorderLayout.WEST);

        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addDoctorFrame.dispose();
                mainFrame.setAlwaysOnTop(true);
                mainFrame.setEnabled(true);
            }
        });

        JButton saveButton = new JButton(SAVE);
```

```
        bottomButtons.add(saveButton, BorderLayout.EAST);


        saveButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    // Checks if all the textfields are filled in.
                    for (int i = 0; i < textFields.size(); i++)
                        if (textFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();

                    manager.getDoctors().add(new Doctor(textFields.get(0).getText(),
textFields.get(1).getText(),
                            textFields.get(2).getText(),
Long.parseLong(textFields.get(3).getText()), textFields.get(4).getText()));
                    searchDoctor.doClick();
                    addDoctorFrame.dispose();
                    mainFrame.setAlwaysOnTop(true);
                    mainFrame.setEnabled(true);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    addDoctorFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addDoctorFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    addDoctorFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addDoctorFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

    }

    /**
     * Creates the frame that is shown when adding a new patient.
     */
    private void createAddPatientDisplay() {
        addPatientFrame = new JFrame();
        addPatientFrame.setAlwaysOnTop(true);
        mainFrame.setEnabled(false);
        mainFrame.setAlwaysOnTop(false);
        addPatientFrame.setLayout(new BorderLayout());

        createAddPatientInformationForm();

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addPatientFrame.dispose();
                addPatientFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };

        addPatientFrame.addWindowListener(exitListener);
        addPatientFrame.setSize(new Dimension(GUI.ADD_PATIENT_FRAME_WIDTH,
GUI.ADD_PATIENT_FRAME_HEIGHT));
        addPatientFrame.setLocationRelativeTo(null);
        addPatientFrame.setVisible(true);
    }

    /**
```

```java
 * Creates the contents in the display shown when adding a new patient. It allows the user to
 * enter information about the patient.
 */
private void createAddPatientInformationForm() {
    JPanel informationFields = new JPanel();
    addPatientFrame.getContentPane().add(informationFields, BorderLayout.CENTER);

    JPanel patientInformation = new JPanel();
    patientInformation.setLayout(new FlowLayout());
    addPatientFrame.getContentPane().add(patientInformation, BorderLayout.NORTH);

    JLabel patientInformationLabel = new JLabel(PATIENT_INFORMATION);
    patientInformation.add(patientInformationLabel);

    Border border = informationFields.getBorder();
    Border margin = new EmptyBorder(20, 20, 20, 20);
    informationFields.setBorder(new CompoundBorder(border, margin));

    informationFields.setLayout(new GridLayout(0, 4));

    ArrayList<JTextField> textFields = new ArrayList<JTextField>();

    String[] labels = {NAME, SEX, SPECIES, BREED, COLOUR, BIRTHDATE,
            WEIGHT};

    // Adds all the textfields to be used by the user to enter information.
    for (int i = 0; i < labels.length; i++) {
        informationFields.add(new JLabel(labels[i]));
        textFields.add(new JTextField(15));
        informationFields.add(textFields.get(i));
    }
    textFields.get(5).setText(DATE_FORMAT_DASH);


    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new BorderLayout());
    addPatientFrame.getContentPane().add(bottomPanel, BorderLayout.SOUTH);

    JPanel ownerListInformation = new JPanel();
    ownerListInformation.setLayout(new BoxLayout(ownerListInformation, BoxLayout.Y_AXIS));

    JComboBox ownerList = new JComboBox(manager.getOwnerList());

    JPanel ownerInfo = new JPanel();
    ownerListInformation.add(ownerInfo);

    // Adds the list from which the user is able to decide whether the information
    // about the owner of this patient is already in the system.
    ownerInfo.add(new JLabel(OWNER));
    ownerInfo.add(ownerList);
    ownerListInformation.add(new JLabel(OWNER_NOT_IN_SYSTEM));

    bottomPanel.add(ownerListInformation, BorderLayout.PAGE_START);

    JPanel bottomButtons = new JPanel();
    bottomButtons.setLayout(new BorderLayout());
    bottomPanel.add(bottomButtons);

    JButton cancelButton = new JButton(CANCEL);
    bottomButtons.add(cancelButton, BorderLayout.WEST);

    cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            addPatientFrame.dispose();
            mainFrame.setAlwaysOnTop(true);
            mainFrame.setEnabled(true);
        }
    });
```

```
        JButton nextButton = new JButton(NEXT);
        bottomButtons.add(nextButton, BorderLayout.EAST);

        nextButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                try {
                    // Checks if all the textfields are filled.
                    for (int i = 0; i < textFields.size(); i++)
                        if (textFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();

                    // Obtains all the information from the textfields
                    String name = textFields.get(0).getText();
                    char sex = textFields.get(1).getText().toCharArray()[0];
                    String species = textFields.get(2).getText();
                    String breed = textFields.get(3).getText();
                    String colour = textFields.get(4).getText();

                    if ((textFields.get(5).getText()).length() != 10)
                        throw new NullPointerException();
                    int dateOfBirth =
Integer.parseInt(manager.changeDateFormat(textFields.get(5).getText(),
                            DATE_FORMAT_DASH, DATE_FORMAT_PLAIN));
                    System.out.println(dateOfBirth);


                    double weight = Double.parseDouble(textFields.get(6).getText());

                    // If the owner is not yet in the system, obtain the owner information.
                    if (ownerList.getSelectedItem().equals("   ")) {
                        addPatientFrame.dispose();
                        createAddOwnerInformationDisplay(new Patient(name, sex, species, breed,
colour,
                                dateOfBirth, weight));

                    } else {
                        // If the owner is already in the system, move on to get information
about the patient staying in a room.
                        String selectedOwner = (String) ownerList.getSelectedItem();
                        createAddPatientRoomOnlyDisplay(new Patient(name, sex, species, breed,
colour, dateOfBirth, weight),
                                selectedOwner);
                    }
                } catch (ArrayIndexOutOfBoundsException e1) {
                    addPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addPatientFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    addPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addPatientFrame.setAlwaysOnTop(true);
                    }
                } catch (NullPointerException e3) {
                    addPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, FORMAT_BIRTHDATE_CORRECT,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addPatientFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });
    }
```

```java
    /**
     * Creates the frame that is shown when adding information about the room in which the given
     * patient stays.
     *
     * @param patient       the patient with only its general information
     * @param selectedOwner the patient's owner name
     */
    private void createAddPatientRoomOnlyDisplay(Patient patient, String selectedOwner) {
        addPatientRoomFrame = new JFrame();
        addPatientRoomFrame.setLayout(new BorderLayout());
        addPatientRoomFrame.setAlwaysOnTop(true);
        if (patient != null) {
            addPatientFrame.setEnabled(false);
            addPatientFrame.setAlwaysOnTop(false);
        } else {
            editPatientFrame.setEnabled(false);
            editPatientFrame.setAlwaysOnTop(false);
        }

        createAddPatientRoomOnlyForm(patient, selectedOwner);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addPatientRoomFrame.dispose();
                addPatientRoomFrame.setAlwaysOnTop(false);
                if (patient != null) {
                    mainFrame.setAlwaysOnTop(true);
                    mainFrame.setEnabled(true);
                } else {
                    editPatientFrame.setEnabled(true);
                    editPatientFrame.setAlwaysOnTop(true);
                }
            }
        };

        addPatientRoomFrame.addWindowListener(exitListener);
        addPatientRoomFrame.setSize(new Dimension(GUI.ADD_PATIENT_ROOM_FRAME_WIDTH,
GUI.ADD_PATIENT_ROOM_FRAME_HEIGHT));
        addPatientRoomFrame.setLocationRelativeTo(null);
        addPatientRoomFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display to add information regarding the room
     * in which the given patient stays.
     *
     * @param newPatient    the patient with only its general information
     * @param selectedOwner the patient's owner's name
     */
    private void createAddPatientRoomOnlyForm(Patient newPatient, String selectedOwner) {
        JPanel titlePanel = new JPanel();
        JPanel contentPanel = new JPanel();
        JPanel bottomPanel = new JPanel();
        addPatientRoomFrame.add(titlePanel, BorderLayout.NORTH);
        addPatientRoomFrame.add(contentPanel, BorderLayout.CENTER);
        addPatientRoomFrame.add(bottomPanel, BorderLayout.SOUTH);

        JLabel roomInformationLabel = new JLabel(ROOM_INFORMATION);
        titlePanel.add(roomInformationLabel);
        roomInformationLabel.setHorizontalAlignment(SwingConstants.CENTER);

        // Obtains the list of all the rooms.
        contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y_AXIS));
        JComboBox roomList = new JComboBox(manager.getReceptionist().getRoomsList());

        JPanel roomListPanel = new JPanel();
        roomListPanel.setLayout(new BoxLayout(roomListPanel, BoxLayout.X_AXIS));
```

```
        contentPanel.add(roomListPanel);


        roomListPanel.add(new JLabel(ROOM));
        roomListPanel.add(roomList);


        JButton cancelButton = new JButton(CANCEL);
        JButton saveButton = new JButton(SAVE);

        bottomPanel.setLayout(new BorderLayout());

        bottomPanel.add(cancelButton, BorderLayout.WEST);
        bottomPanel.add(saveButton, BorderLayout.EAST);

        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addPatientRoomFrame.dispose();
                if (newPatient != null) {
                    addPatientFrame.setEnabled(true);
                    addPatientFrame.setAlwaysOnTop(true);
                    mainFrame.setAlwaysOnTop(false);
                    mainFrame.setEnabled(false);
                } else {
                    editPatientFrame.setAlwaysOnTop(true);
                    editPatientFrame.setEnabled(true);
                }

            }
        });

        saveButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {


                    String selectedRoomOption = (String) roomList.getSelectedItem();
                    boolean found = false;
                    Room room = null;
                    boolean noRoom = false;

                    if (selectedRoomOption.equals(ASSIGN_ROOM_AUTOMATICALLY)) {
                        for (int i = 0; i < manager.getReceptionist().getRooms().size() &&
!found; i++) {
                            if (!manager.getReceptionist().getRooms().get(i).isFull()) {
                                room = manager.getReceptionist().getRooms().get(i);
                                found = true;
                            }
                        }
                        if (!found) {
                            throw new ArrayIndexOutOfBoundsException();
                        }

                    } else if (selectedRoomOption.equals(NONE)) {
                        noRoom = true;
                    } else {
                        room =
manager.getReceptionist().getRooms().get(roomList.getSelectedIndex() - 2);
                    }

                    if (newPatient != null) {
                        Patient patient = getSameOwnerInformation(selectedOwner);
                        manager.getPatients().add(newPatient);
                        manager.getPatients().get(manager.getPatients().size() -
1).addOwnerInformation(
                                patient.getOwnerFirstName(), patient.getOwnerLastName(),
patient.getOwnerEmail(),
                                patient.getOwnerNumber(), patient.getOwnerAddress());
```

```
                        if (noRoom) {
                            manager.getPatients().get(manager.getPatients().size() -
1).setAssignedRoom(NONE);
                        } else {

room.addPatientToRoom(manager.getPatients().get(manager.getPatients().size() - 1));
                            manager.getPatients().get(manager.getPatients().size() -
1).setAssignedRoom(room.getName());
                            manager.getPatients().get(manager.getPatients().size() -
1).setDateOfRoomAdmittance(manager.returnDateToday());

                        }
                        addPatientFrame.dispose();
                        searchPatient.doClick();
                        addPatientRoomFrame.dispose();
                        mainFrame.setAlwaysOnTop(true);
                        mainFrame.setEnabled(true);
                    } else {
                        if (noRoom) {
                            manager.findPatient(selectedOwner).setAssignedRoom(NONE);
                        } else {
                            manager.findPatient(selectedOwner).setAssignedRoom(room.getName());

manager.findPatient(selectedOwner).setDateOfRoomAdmittance(manager.returnDateToday());
                            room.addPatientToRoom(manager.findPatient(selectedOwner));
                        }

                        addPatientRoomFrame.dispose();
                        editPatientFrame.dispose();
                        createEditPatientDisplay(selectedOwner);
                        editPatientFrame.setAlwaysOnTop(true);
                        editPatientFrame.setEnabled(true);
                    }


                } catch (NullPointerException e1) {
                    if (newPatient != null) {
                        addPatientRoomFrame.setAlwaysOnTop(false);
                        int result = JOptionPane.showConfirmDialog(null, ERROR, ERROR,
JOptionPane.DEFAULT_OPTION);
                        if (result == JOptionPane.OK_OPTION) {
                            addPatientRoomFrame.setAlwaysOnTop(true);
                        }
                    } else {
                        editPatientFrame.setAlwaysOnTop(false);
                        int result = JOptionPane.showConfirmDialog(null, ERROR, ERROR,
JOptionPane.DEFAULT_OPTION);
                        if (result == JOptionPane.OK_OPTION) {
                            editPatientFrame.setAlwaysOnTop(true);
                        }
                    }
                } catch (ArrayIndexOutOfBoundsException e2) {
                    if (newPatient != null) {
                        addPatientRoomFrame.setAlwaysOnTop(false);
                        int result = JOptionPane.showConfirmDialog(null, NO_ROOMS_AVAILIABLE,
ERROR, JOptionPane.DEFAULT_OPTION);
                        if (result == JOptionPane.OK_OPTION) {
                            addPatientRoomFrame.setAlwaysOnTop(true);
                        }
                    } else {
                        editPatientFrame.setAlwaysOnTop(false);
                        int result = JOptionPane.showConfirmDialog(null, NO_ROOMS_AVAILIABLE,
ERROR, JOptionPane.DEFAULT_OPTION);
                        if (result == JOptionPane.OK_OPTION) {
                            editPatientFrame.setAlwaysOnTop(true);
                        }
                    }
```

```
                }


            }
        });

        roomListPanel.setBorder(new CompoundBorder(roomListPanel.getBorder(), new EmptyBorder(0,
10, 0, 0)));
    }


    /**
     * Using the specified first and last name of an owner,
     * a patient belonging to the owner is found and returned.
     *
     * @param name the first and last name of the specified owner
     * @return a patient with the same owner
     */
    private Patient getSameOwnerInformation(String name) {
        for (int i = 0; i < manager.getPatients().size(); i++)
            if ((manager.getPatients().get(i).getOwnerFirstName() + " "
                    + manager.getPatients().get(i).getOwnerLastName()).equals(name))
                return manager.getPatients().get(i);

        return null;
    }

    /**
     * Creates the frame that is shown when adding information about the owner
     * of a specified patient.
     *
     * @param patient the patient whose owner information is being added
     */
    private void createAddOwnerInformationDisplay(Patient patient) {
        addPatientFrame = new JFrame();
        addPatientFrame.setLayout(new BorderLayout());
        addPatientFrame.setAlwaysOnTop(true);

        createAddOwnerInformationForm(patient);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addPatientFrame.dispose();
                addPatientFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };

        addPatientFrame.addWindowListener(exitListener);
        addPatientFrame.setSize(new Dimension(GUI.ADD_PATIENT_OWNER_FRAME_WIDTH,
GUI.ADD_PATIENT_OWNER_FRAME_HEIGHT));
        addPatientFrame.setLocationRelativeTo(null);
        addPatientFrame.setResizable(false);
        addPatientFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display for the user to input the information about the owner
     * of the specified patient.
     *
     * @param patient the patient whose owner information is being added
     */
    private void createAddOwnerInformationForm(Patient patient) {
        JPanel informationFields = new JPanel();
        addPatientFrame.getContentPane().add(informationFields, BorderLayout.CENTER);

        JPanel patientInformation = new JPanel();
```

```
patientInformation.setLayout(new FlowLayout());
addPatientFrame.getContentPane().add(patientInformation, BorderLayout.NORTH);

JLabel ownerInformationLabel = new JLabel(OWNER_INFORMATION);
patientInformation.add(ownerInformationLabel);

Border border = informationFields.getBorder();
Border margin = new EmptyBorder(20, 20, 20, 20);
informationFields.setBorder(new CompoundBorder(border, margin));

informationFields.setLayout(new GridLayout(0, 4));
ArrayList<JTextField> textFields = new ArrayList<JTextField>();

String[] labels = {FIRST_NAME, LAST_NAME, EMAIL, TELEPHONE_NUMBER, ADDRESS};

// Adds textfields used to enter the owner's information.
for (int i = 0; i < labels.length; i++) {
    informationFields.add(new JLabel(labels[i]));
    textFields.add(new JTextField(15));
    informationFields.add(textFields.get(i));
}

JPanel bottomPanel = new JPanel();
bottomPanel.setLayout(new BorderLayout());
addPatientFrame.getContentPane().add(bottomPanel, BorderLayout.SOUTH);

JPanel ownerListInformation = new JPanel();
ownerListInformation.setLayout(new FlowLayout());


JPanel roomListPanel = new JPanel();
roomListPanel.setLayout(new BoxLayout(roomListPanel, BoxLayout.X_AXIS));

ownerListInformation.add(roomListPanel);

JComboBox roomList = new JComboBox(manager.getReceptionist().getRoomsList());

roomListPanel.add(new JLabel(ROOM));
roomListPanel.add(roomList);

bottomPanel.add(ownerListInformation, BorderLayout.PAGE_START);

JPanel bottomButtons = new JPanel();
bottomButtons.setLayout(new BorderLayout());
bottomPanel.add(bottomButtons);

JButton cancelButton = new JButton(CANCEL);
bottomButtons.add(cancelButton, BorderLayout.WEST);

cancelButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        addPatientFrame.dispose();
        mainFrame.setAlwaysOnTop(true);
        mainFrame.setEnabled(true);
    }
});

JButton finishButton = new JButton(FINISH);
bottomButtons.add(finishButton, BorderLayout.EAST);

finishButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            // Checks if all the textfields are filled.
            for (int i = 0; i < textFields.size(); i++)
                if (textFields.get(i).getText().trim().equals(""))
                    throw new ArrayIndexOutOfBoundsException();

            // Obtains owner's information from the textfields.
```

```
                    String ownerFirstName = textFields.get(0).getText();
                    String ownerLastName = textFields.get(1).getText();
                    String ownerEmail = textFields.get(2).getText();
                    long ownerNumber = Long.parseLong(textFields.get(3).getText());
                    String ownerAddress = textFields.get(4).getText();

                    String selectedRoomOption = (String) roomList.getSelectedItem();
                    Room room = null;
                    boolean found = false;
                    boolean noRoom = false;

                    if (selectedRoomOption.equals(ASSIGN_ROOM_AUTOMATICALLY)) {
                        // Finds a room that has space availiable and assigns it to the specified
patient.
                        for (int i = 0; i < manager.getReceptionist().getRooms().size() &&
!found; i++) {
                            if (!manager.getReceptionist().getRooms().get(i).isFull()) {
                                room = manager.getReceptionist().getRooms().get(i);
                                found = true;
                            }
                        }
                        if (!found)
                            throw new NullPointerException();


                    } else if (selectedRoomOption.equals(NONE)) {
                        noRoom = true;

                    } else {
                        room =
manager.getReceptionist().getRooms().get(roomList.getSelectedIndex() - 2);

                    }

                    manager.getPatients().add(patient);
                    manager.getPatients().get(manager.getPatients().size() -
1).addOwnerInformation(
                            ownerFirstName, ownerLastName, ownerEmail,
                            ownerNumber, ownerAddress);
                    if (noRoom) {
                        manager.getPatients().get(manager.getPatients().size() -
1).setAssignedRoom(NONE);
                    } else {

room.addPatientToRoom(manager.getPatients().get(manager.getPatients().size() - 1));
                        manager.getPatients().get(manager.getPatients().size() -
1).setAssignedRoom(room.getName());
                        manager.getPatients().get(manager.getPatients().size() -
1).setDateOfRoomAdmittance(manager.returnDateToday());
                    }


                    searchPatient.doClick();
                    addPatientFrame.dispose();
                    mainFrame.setAlwaysOnTop(true);
                    mainFrame.setEnabled(true);

                } catch (ArrayIndexOutOfBoundsException e1) {
                    addPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addPatientFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    addPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
```

```
                    addPatientFrame.setAlwaysOnTop(true);
                }
            } catch (NullPointerException e3) {
                addPatientFrame.setAlwaysOnTop(false);
                int result = JOptionPane.showConfirmDialog(null, NO_ROOMS_AVAILIABLE, ERROR,
JOptionPane.DEFAULT_OPTION);
                if (result == JOptionPane.OK_OPTION) {
                    addPatientFrame.setAlwaysOnTop(true);
                }
            }
        }
    });

}

/**
 * Creates the frame displaying information about the specified patient. It includes the
patient's
 * general information, owner information, appointments, past examinations and the
laboaratory tests
 * performed.
 *
 * @param uniqueID the unique ID of the specified patient
 */
private void createEditPatientDisplay(String uniqueID) {
    mainFrame.setEnabled(false);
    mainFrame.setAlwaysOnTop(false);
    editPatientFrame = new JFrame();
    editPatientFrame.setAlwaysOnTop(true);
    createEditPatientForm(uniqueID);

    WindowListener exitListener = new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            editPatientFrame.dispose();
            editPatientFrame.setAlwaysOnTop(false);
            mainFrame.setEnabled(true);
        }
    };

    editPatientFrame.addWindowListener(exitListener);
    editPatientFrame.setSize(new Dimension(GUI.EDIT_PATIENT_FRAME_WIDTH,
GUI.EDIT_PATIENT_FRAME_HEIGHT));
    editPatientFrame.setLocationRelativeTo(null);
    editPatientFrame.setVisible(true);
}

/**
 * Creates the contents of the display showing information about the specified patient.
 * It includes the patient's general information, owner information, appointments,
 * past examinations and the laboaratory tests performed.
 *
 * @param uniqueID the unique ID of the specified patient
 */
private void createEditPatientForm(String uniqueID) {
    JPanel leftEditPanel = new JPanel();
    JPanel mainPanel = new JPanel();
    editPatientFrame.getContentPane().add(mainPanel);

    mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.X_AXIS));
    mainPanel.add(leftEditPanel);

    leftEditPanel.setLayout(new BoxLayout(leftEditPanel, BoxLayout.Y_AXIS));

    JPanel generalInfo = new JPanel();
    generalInfo.setLayout(new FlowLayout());
    JLabel generalInformationLabel = new JLabel(GENERAL_INFORMATION);
    generalInfo.add(generalInformationLabel);
    JButton editGI = new JButton(EDIT);
```

```
        generalInfo.add(editGI);

        editGI.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                editPatientFrame.setEnabled(false);
                editPatientFrame.setAlwaysOnTop(false);
                createEditPatientGeneralInfoDisplay(uniqueID);
            }
        });

        leftEditPanel.add(generalInfo);
        generalInformationLabel.setHorizontalAlignment(SwingConstants.CENTER);

        JPanel generalPatientInformationEdit = new JPanel();
        leftEditPanel.add(generalPatientInformationEdit);
        generalPatientInformationEdit.setLayout(new GridLayout(0, 4));

        ArrayList<JLabel> textFields = new ArrayList<JLabel>();

        String[] labels1 = {NAME, SEX, SPECIES, BREED, COLOUR, BIRTHDATE,
                WEIGHT};

        Patient selectedPatient = manager.findPatient(uniqueID);

        String[] defaultInfo = {selectedPatient.getName(), selectedPatient.getSex() + "",
selectedPatient.getSpecies(),
                selectedPatient.getBreed(), selectedPatient.getColour(),
manager.changeDateFormat(
                selectedPatient.getDateOfBirth() + "", DATE_FORMAT_PLAIN, DATE_FORMAT_DASH),
                selectedPatient.getWeight() + ""};

        JLabel label;

        // Displays general information about this patient.
        for (int i = 0; i < labels1.length; i++) {
            label = new JLabel(labels1[i]);
            label.setFont(new Font(label.getFont().getName(),
                    Font.BOLD, label.getFont().getSize()));
            generalPatientInformationEdit.add(label);
            textFields.add(new JLabel(defaultInfo[i]));
            generalPatientInformationEdit.add(textFields.get(i));
        }
        label = new JLabel(ASSIGNED_ROOM);
        label.setFont(new Font(label.getFont().getName(),
                Font.BOLD, label.getFont().getSize()));

        JPanel roomInfo = new JPanel();
        leftEditPanel.add(roomInfo);

        JButton removePatientButton = new JButton("Remove Patient");
        roomInfo.add(removePatientButton);

        removePatientButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editPatientFrame.setAlwaysOnTop(false);
                int result = JOptionPane.showConfirmDialog(null, "Are you sure that you would
like to remove this patient from the system?", ERROR, JOptionPane.YES_NO_OPTION);
                if (result == JOptionPane.YES_OPTION) {
                    editPatientFrame.setAlwaysOnTop(true);
                    for(int i = 0; i < manager.getReceptionist().getAppointments().size();i++)
                    {

if(manager.getReceptionist().getAppointments().get(i).getPatient().getUniqueID().equals(uniqueID)
)
                        {
                            for(int a = 0; a < manager.getDoctors().size();a++)
                                for(int b = 0; b <
manager.getDoctors().get(a).getTasks().size();b++)
```

```
if(manager.getDoctors().get(a).getTasks().get(b).getUniqueID().equals

(manager.getReceptionist().getAppointments().get(i).getUniqueID()))
                                    manager.getDoctors().get(a).getTasks().remove(b);
                            manager.getReceptionist().getAppointments().remove(i);
                        }

                            manager.getPatients().remove(manager.findPatient(uniqueID));
                    }
                    editPatientFrame.dispose();
                    searchPatient.doClick();
                    mainFrame.setEnabled(true);
                    mainFrame.setAlwaysOnTop(true);
                }
                else if(result == JOptionPane.NO_OPTION)
                {
                    editPatientFrame.setAlwaysOnTop(true);
                }

            }
        });

        roomInfo.add(label);

        if (selectedPatient.getAssignedRoom().equals(NONE)) {
            roomInfo.add(new JLabel(NONE));
            JButton assignRoomButton = new JButton(ASSIGN_ROOM);
            roomInfo.add(assignRoomButton);
            assignRoomButton.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    createAddPatientRoomOnlyDisplay(null, uniqueID);
                }
            });
        } else {
            roomInfo.add(new JLabel(selectedPatient.getAssignedRoom()));
            JButton removePatientFromRoomButton = new JButton(GENERATE_BILL);
            roomInfo.add(removePatientFromRoomButton);

            removePatientFromRoomButton.addActionListener(new ActionListener() {
                @Override
                public void actionPerformed(ActionEvent e) {
                    editPatientFrame.setAlwaysOnTop(false);
                    editPatientFrame.setEnabled(false);
                    int result = JOptionPane.showConfirmDialog(null, TOTAL_COST_FOR_ROOM +
manager.getAccountant().generateRoomCharge(
                            manager.findPatient(uniqueID), manager.findRoom(
                                    manager.findPatient(uniqueID).getAssignedRoom())
                    ) + "$", COST, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        manager.findRoom(manager.findPatient(uniqueID).

getAssignedRoom()).removePatientFromRoom(manager.findPatient(uniqueID));
                        manager.findPatient(uniqueID).setAssignedRoom(NONE);
                        manager.findPatient(uniqueID).setDateOfRoomAdmittance(0);
                        editPatientFrame.dispose();
                        createEditPatientDisplay(uniqueID);
                        editPatientFrame.setAlwaysOnTop(true);
                        editPatientFrame.setEnabled(true);
                    }


                }
            });
        }


        Border border = generalPatientInformationEdit.getBorder();
```

```
        Border margin = new EmptyBorder(20, 20, 20, 20);
        generalPatientInformationEdit.setBorder(new CompoundBorder(border, margin));

        // Displays information about the patient's owner.
        JPanel ownerInformation = new JPanel();
        ownerInformation.setLayout(new FlowLayout());
        JLabel ownerInformationLabel = new JLabel(OWNER_INFORMATION);
        ownerInformation.add(ownerInformationLabel);
        JButton editOI = new JButton(EDIT);
        ownerInformation.add(editOI);
        leftEditPanel.add(ownerInformation);
        ownerInformationLabel.setHorizontalAlignment(SwingConstants.CENTER);

        editOI.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createEditOwnerInfoDisplay(uniqueID);
                editPatientFrame.setEnabled(false);
                editPatientFrame.setAlwaysOnTop(false);
            }


        });

        JPanel generalOwnerInformationEdit = new JPanel();
        leftEditPanel.add(generalOwnerInformationEdit);
        generalOwnerInformationEdit.setLayout(new GridLayout(0, 4));

        ArrayList<JLabel> infoLabels2 = new ArrayList<JLabel>();

        String[] labels2 = {FIRST_NAME, LAST_NAME, EMAIL, TELEPHONE_NUMBER, ADDRESS};

        String address = selectedPatient.getOwnerAddress();
        if(selectedPatient.getOwnerAddress().length() > 15)
            address = selectedPatient.getOwnerAddress().substring(0,15) + "...";
        String[] defaultInfo2 = {selectedPatient.getOwnerFirstName(),
selectedPatient.getOwnerLastName(),
                selectedPatient.getOwnerEmail()+" ", selectedPatient.getOwnerNumber() + "",
                address};

        for (int i = 0; i < labels2.length; i++) {
            label = new JLabel(labels2[i]);
            label.setFont(new Font(label.getFont().getName(), Font.BOLD,
label.getFont().getSize()));
            generalOwnerInformationEdit.add(label);
            infoLabels2.add(new JLabel(defaultInfo2[i]));
            generalOwnerInformationEdit.add(infoLabels2.get(i));
        }

        generalOwnerInformationEdit.setBorder(new CompoundBorder(border, margin));

        // Displays information about this patient's upcoming appointments.
        JPanel upcomingAppointments = new JPanel();
        upcomingAppointments.setLayout(new BoxLayout(upcomingAppointments, BoxLayout.Y_AXIS));

        JPanel upcomingAppointmentsHeader = new JPanel();
        upcomingAppointments.add(upcomingAppointmentsHeader);
        upcomingAppointmentsHeader.add(new JLabel(UPCOMING_APPOINTMENTS));

        JButton addAppointment = new JButton(ADD);
        upcomingAppointmentsHeader.add(addAppointment);

        JButton removeAppointment = new JButton(REMOVE);
        upcomingAppointmentsHeader.add(removeAppointment);


        addAppointment.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                editPatientFrame.setEnabled(false);
                editPatientFrame.setAlwaysOnTop(false);
```

```
                createAddAppointmentDisplay(uniqueID);

            }
        });

        leftEditPanel.add(upcomingAppointments);


        // Show Appointments with an option of adding and canceling
        ArrayList<Appointment> appointment =
manager.getReceptionist().getAppointments(selectedPatient);

        String[][] appointmentData = new String[appointment.size()][3];

        for (int i = 0; i < appointmentData.length; i++) {
            appointmentData[i][0] = appointment.get(i).getPurpose() + "";

            appointmentData[i][1] = manager.changeDateFormat(appointment.get(i).getDate() + "",
DATE_FORMAT_PLAIN, DATE_FORMAT_DASH);

            appointmentData[i][2] =
manager.returnFormattedTime(appointment.get(i).getStartTime(),
                    appointment.get(i).getEndTime());

        }

        JTable appointmentTable = new JTable(appointmentData, appointmentColumns);
        JScrollPane appointmentPane = new JScrollPane(appointmentTable);
        appointmentTable.setDefaultEditor(Object.class, null);

        upcomingAppointments.add(appointmentPane);

        upcomingAppointments
                .setBorder(new CompoundBorder(upcomingAppointments.getBorder(), new
EmptyBorder(20, 20, 20, 20)));

        removeAppointment.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    Appointment appointmentToBeDeleted =
manager.getReceptionist().getAppointments(

manager.findPatient(uniqueID)).get(appointmentTable.getSelectedRow());
                    Doctor doctor = appointmentToBeDeleted.getDoctor();
                    for (int a = 0; a < doctor.getTasks().size(); a++) {
                        if
(doctor.getTasks().get(a).getUniqueID().equals(appointmentToBeDeleted.getUniqueID()))
                            doctor.getTasks().remove(a);
                    }
                    manager.getReceptionist().getAppointments().remove(appointmentToBeDeleted);
                    editPatientFrame.dispose();
                    createEditPatientDisplay(uniqueID);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, SELECT_APPOINTMENT_FIRST,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editPatientFrame.setAlwaysOnTop(true);
                    }
                }

            }
        });


        JPanel rightPanel = new JPanel();

        rightPanel.setLayout(new BoxLayout(rightPanel, BoxLayout.Y_AXIS));
```

```
        JPanel history = new JPanel();
        history.setLayout(new BoxLayout(history, BoxLayout.Y_AXIS));

        mainPanel.add(rightPanel);

        rightPanel.add(history);

        JPanel historyTop = new JPanel();

        history.add(historyTop);

        // Displays information about past appointments.

        historyTop.add(new JLabel(HISTORY));

        JButton addHistoryButton = new JButton(ADD);
        historyTop.add(addHistoryButton);

        JButton viewHistoryButton = new JButton(VIEW);
        historyTop.add(viewHistoryButton);

        addHistoryButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createEditHistoryExaminationDisplay(uniqueID);
                editPatientFrame.setEnabled(false);
                editPatientFrame.setAlwaysOnTop(false);
            }
        });


        ArrayList<Examination> examinations =
manager.findPatient(uniqueID).getRecord().getPastExaminations();

        String[][] historyData = new String[examinations.size()][5];

        JPanel[][] area = new JPanel[historyData.length][5];

        for (int i = 0; i < examinations.size(); i++) {
            int reverse = examinations.size() - i - 1;
            historyData[reverse][0] = manager.changeDateFormat(examinations.get(i).getDate() +
"", DATE_FORMAT_PLAIN, DATE_FORMAT_DASH);
            historyData[reverse][1] = examinations.get(i).getSymptom();
            historyData[reverse][2] = examinations.get(i).getDiagnosis();
            historyData[reverse][3] = examinations.get(i).getTreatment();
            historyData[reverse][4] = examinations.get(i).getRemarks();

            for (int a = 0; a < 5; a++) {
                area[i][a] = new JPanel();
                area[i][a].add(new JLabel(historyData[i][a]));
            }
        }

        JTable historyTable = new JTable(historyData, historyColumns);
        JScrollPane historyPane = new JScrollPane(historyTable);
        historyTable.setDefaultEditor(Object.class, null);

        viewHistoryButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int index;
                try {
                    index = historyTable.getSelectedRow();
                    if (index == -1)
                        throw new ArrayIndexOutOfBoundsException();
                    createViewHistoryExaminationDisplay(uniqueID, index);
                    editPatientFrame.setEnabled(false);
                    editPatientFrame.setAlwaysOnTop(false);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editPatientFrame.setAlwaysOnTop(false);
```

```
                    int result = JOptionPane.showConfirmDialog(null, SELECT_EXAMINATION_FIRST,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editPatientFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

        history.add(historyPane);

        history.setBorder(new CompoundBorder(history.getBorder(), new EmptyBorder(20, 20, 20,
20)));


        JPanel labReport = new JPanel();
        labReport.setLayout(new BoxLayout(labReport, BoxLayout.Y_AXIS));

        rightPanel.add(labReport);


        String[][] labData = new String[manager.findPatient(uniqueID).getTests().size()][4];

        Test test;

        for (int i = 0; i < labData.length; i++) {
            test = manager.findPatient(uniqueID).getTests().get(i);
            labData[i][0] = test.getTestType();
            labData[i][1] = manager.changeDateFormat(test.getDate() + "",
                    DATE_FORMAT_PLAIN, DATE_FORMAT_DASH);
            labData[i][2] = test.getStatus();
        }

        // Displays information about laboratory tests performed with this patient.
        JTable labTable = new JTable(labData, labColumns);
        JScrollPane labPane = new JScrollPane(labTable);
        labTable.setDefaultEditor(Object.class, null);

        JPanel laboratoryTestTop = new JPanel();

        labReport.add(laboratoryTestTop);
        laboratoryTestTop.add(new JLabel(LABORATORY_TESTS));

        JButton addLabTestButton = new JButton(ADD);
        laboratoryTestTop.add(addLabTestButton);

        JButton changeStatusButton = new JButton(CHANGE_STATUS);
        laboratoryTestTop.add(changeStatusButton);

        changeStatusButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    int selectedRow = labTable.getSelectedRow();
                    manager.findPatient(uniqueID).getTests().get(selectedRow).changeStatus();
                    editPatientFrame.dispose();
                    createEditPatientDisplay(uniqueID);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null,
SELECT_LABORATORY_TEST_FIRST, ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editPatientFrame.setAlwaysOnTop(true);
                    }
                } catch (NullPointerException e2) {
                    editPatientFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, STATUS_CANNOT_BE_CHANGED,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
```

```
                    editPatientFrame.setAlwaysOnTop(true);
                }

            }
        }
    });


    JButton removeLabTestButton = new JButton(REMOVE);
    laboratoryTestTop.add(removeLabTestButton);

    addLabTestButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            editPatientFrame.setEnabled(false);
            editPatientFrame.setAlwaysOnTop(false);
            createEditPatientLaboratoryTestsDisplay(uniqueID);
        }
    });

    removeLabTestButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                int selectedRow = labTable.getSelectedRow();
                manager.findPatient(uniqueID).getTests().remove(selectedRow);
                editPatientFrame.dispose();
                createEditPatientDisplay(uniqueID);
            } catch (ArrayIndexOutOfBoundsException e1) {
                editPatientFrame.setAlwaysOnTop(false);
                int result = JOptionPane.showConfirmDialog(null,
SELECT_LABORATORY_TEST_FIRST, ERROR, JOptionPane.DEFAULT_OPTION);
                if (result == JOptionPane.OK_OPTION) {
                    editPatientFrame.setAlwaysOnTop(true);
                }
            }

        }
    });

    labReport.add(labPane);

    labReport.setBorder(new CompoundBorder(labReport.getBorder(), new EmptyBorder(20, 20, 20,
20)));

}

/**
 * Creates the frame that displays an examination performed with this patient performed in
the past.
 * The index specifies the examination and the unique ID specifies the patient.
 *
 * @param uniqueID the unique ID of the patient
 * @param index    the index of the exmaination performed
 */
private void createViewHistoryExaminationDisplay(String uniqueID, int index) {
    displayHistoryFrame = new JFrame();
    displayHistoryFrame.setAlwaysOnTop(true);

    createViewHistoryExaminationForm(uniqueID, index);

    WindowListener exitListener = new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
            displayHistoryFrame.dispose();
            displayHistoryFrame.setAlwaysOnTop(false);
            editPatientFrame.setEnabled(true);
            editPatientFrame.setAlwaysOnTop(true);
        }
    };
    displayHistoryFrame.addWindowListener(exitListener);
```

```
        displayHistoryFrame
                .setSize(new Dimension(GUI.DISPLAY_HISTORY_FRAME_WIDTH,
GUI.DISPLAY_HISTORY_FRAME_HEIGHT));
        displayHistoryFrame.setLocationRelativeTo(null);
        displayHistoryFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display showing an examination performed with this patient
performed in the past.
     * The index specifies the examination and the unique ID specifies the patient.
     *
     * @param uniqueID the unique ID of the patient
     * @param index    the index of the exmaination performed
     */
    private void createViewHistoryExaminationForm(String uniqueID, int index) {

        JPanel totalPanel = new JPanel();
        displayHistoryFrame.getContentPane().add(totalPanel);

        totalPanel.setLayout(new BoxLayout(totalPanel, BoxLayout.Y_AXIS));

        totalPanel.add(new JLabel(EXAMINATION_INFORMATION));

        Examination examination =
manager.findPatient(uniqueID).getRecord().getPastExaminations().get(index);

        JPanel dateInfo = new JPanel();
        dateInfo.add(new JLabel(DATE + manager.changeDateFormat(examination.getDate() + "",
DATE_FORMAT_PLAIN, DATE_FORMAT_DASH)));

        JPanel symptomPanel = new JPanel();
        totalPanel.add(symptomPanel);

        symptomPanel.add(new JLabel(SYMPTOMS));
        JTextArea symptomArea = new JTextArea(8, 40);
        symptomPanel.add(symptomArea);
        symptomArea.setText(examination.getSymptom());
        symptomArea.setEditable(false);

        JPanel diagnosisPanel = new JPanel();
        totalPanel.add(diagnosisPanel);

        diagnosisPanel.add(new JLabel(DIAGNOSIS));
        JTextArea diagnosisArea = new JTextArea(8, 40);
        diagnosisPanel.add(diagnosisArea);
        diagnosisArea.setText(examination.getDiagnosis());
        diagnosisArea.setEditable(false);

        JPanel treatmentPanel = new JPanel();
        totalPanel.add(treatmentPanel);

        treatmentPanel.add(new JLabel(TREATMENT));
        JTextArea treatmentArea = new JTextArea(8, 40);
        treatmentPanel.add(treatmentArea);
        treatmentArea.setText(examination.getTreatment());
        treatmentArea.setEditable(false);

        JPanel remarksPanel = new JPanel();
        totalPanel.add(remarksPanel);

        remarksPanel.add(new JLabel(REMARKS));
        JTextArea remarksArea = new JTextArea(8, 40);
        remarksPanel.add(remarksArea);
        remarksArea.setText(examination.getRemarks());
        remarksArea.setEditable(false);

    }
```

```
    /**
     * Creates the frame that displayed the specified patient's laboratory tests that will soon
be performed
     * or have been performed.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditPatientLaboratoryTestsDisplay(String uniqueID) {
        addPatientLabTestFrame = new JFrame();
        addPatientLabTestFrame.setAlwaysOnTop(true);

        createEditPatientLaboratoryTestsForm(uniqueID);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addPatientLabTestFrame.dispose();
                addPatientLabTestFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        };
        addPatientLabTestFrame.addWindowListener(exitListener);
        addPatientLabTestFrame
                .setSize(new Dimension(GUI.ADD_LAB_TEST_FRAME_WIDTH,
GUI.ADD_LAB_TEST_FRAME_HEIGHT));
        addPatientLabTestFrame.setLocationRelativeTo(null);
        addPatientLabTestFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display showing the specified patient's laboratory tests that
will soon be performed
     * or have been performed.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditPatientLaboratoryTestsForm(String uniqueID) {
        JPanel totalPanel = new JPanel();
        addPatientLabTestFrame.getContentPane().add(totalPanel);

        totalPanel.setLayout(new BoxLayout(totalPanel, BoxLayout.Y_AXIS));

        JPanel titlePanel = new JPanel();
        titlePanel.add(new JLabel(LABORATORY_TEST_INFORMATION));
        totalPanel.add(titlePanel);

        JPanel testType = new JPanel();
        totalPanel.add(testType);

        testType.add(new JLabel(TEST_TYPE));
        testTypes = manager.getLabTestsList();
        JComboBox testTypeSelect = new JComboBox(testTypes);
        testType.add(testTypeSelect);

        JPanel status = new JPanel();
        totalPanel.add(status);

        status.add(new JLabel(STATUS));
        JComboBox statusSelect = new JComboBox(statusList);
        status.add(statusSelect);

        JPanel dateInformation = new JPanel();
        totalPanel.add(dateInformation);

        JPanel yearInformation = new JPanel();
        dateInformation.add(yearInformation);

        yearInformation.add(new JLabel(YEAR));
```

```
String[] years = {Calendar.getInstance().get(Calendar.YEAR) + "",
        (Calendar.getInstance().get(Calendar.YEAR) + 1) + ""};

JComboBox yearChoice = new JComboBox(years);
yearInformation.add((yearChoice));

JPanel dayMonthInformation = new JPanel();
dateInformation.add(dayMonthInformation);
dayMonthInformation.setLayout(new BoxLayout(dayMonthInformation, BoxLayout.X_AXIS));


dayMonthInformation.add(new JLabel(MONTH));
JComboBox monthSelect = new JComboBox(months);


dayMonthInformation.add(monthSelect);

int numberOfDaysInMonth = 31;

String[] days = new String[numberOfDaysInMonth];

for (int i = 0; i < numberOfDaysInMonth; i++) {
    days[i] = i + 1 + "";
}

JComboBox daySelect = new JComboBox(days);

dayMonthInformation.add(new JLabel(DAY));
dayMonthInformation.add(daySelect);

monthSelect.setSelectedIndex(Calendar.getInstance().get(Calendar.MONTH));
daySelect.setSelectedIndex(Calendar.getInstance().get(Calendar.DAY_OF_MONTH) - 1);

JPanel bottomSection = new JPanel();
totalPanel.add(bottomSection);

bottomSection.setLayout(new BorderLayout());

JButton cancelButton = new JButton(CANCEL);
JButton saveButton = new JButton(SAVE);

bottomSection.add(cancelButton, BorderLayout.WEST);
bottomSection.add(saveButton, BorderLayout.EAST);

saveButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            String testType = (String) testTypeSelect.getSelectedItem();
            if(testType == null)
                throw new NullPointerException();
            String status = (String) statusSelect.getSelectedItem();
            int date;

            String day = "" + daySelect.getSelectedItem();

            String month;

            if (monthSelect.getSelectedIndex() + 1 > 9)
                month = "" + (monthSelect.getSelectedIndex() + 1);
            else
                month = "0" + (monthSelect.getSelectedIndex() + 1);

            String year = "" + yearChoice.getSelectedItem();

            date = Integer.parseInt(day + month + year);


            manager.findPatient(uniqueID).getTests().add(new Test(testType, status,
```

```
date));

                    addPatientLabTestFrame.dispose();
                    editPatientFrame.dispose();
                    createEditPatientDisplay(uniqueID);
                    addPatientLabTestFrame.setAlwaysOnTop(false);
                    editPatientFrame.setEnabled(true);
                    editPatientFrame.setAlwaysOnTop(true);
                }catch (NullPointerException e1)
                {
                    addPatientLabTestFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null,
SELECT_LABORATORY_TEST_FIRST, ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addPatientLabTestFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addPatientLabTestFrame.dispose();
                addPatientLabTestFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        });
    }


    /**
     * Creates the frame used to add a new general examination information for the specified
patient.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditHistoryExaminationDisplay(String uniqueID) {
        addPatientExaminationFrame = new JFrame();
        addPatientExaminationFrame.setAlwaysOnTop(true);

        createEditHistoryExaminationForm(uniqueID);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addPatientExaminationFrame.dispose();
                addPatientExaminationFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        };
        addPatientExaminationFrame.addWindowListener(exitListener);
        addPatientExaminationFrame
                .setSize(new Dimension(GUI.EDIT_PATIENT_FRAME_WIDTH - 450,
GUI.EDIT_PATIENT_FRAME_HEIGHT + 100));
        addPatientExaminationFrame.setLocationRelativeTo(null);
        addPatientExaminationFrame.setVisible(true);

    }

    /**
     * Creates the contents of the display shown when adding a new general examination
     * information for the specified patient.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditHistoryExaminationForm(String uniqueID) {
        JPanel totalPanel = new JPanel();
```

```
addPatientExaminationFrame.getContentPane().add(totalPanel);

totalPanel.setLayout(new BoxLayout(totalPanel, BoxLayout.Y_AXIS));

totalPanel.add(new JLabel(EXAMINATION_INFORMATION));

JPanel dateInformation = new JPanel();
totalPanel.add(dateInformation);

JPanel yearInformation = new JPanel();
dateInformation.add(yearInformation);

yearInformation.add(new JLabel(YEAR));

String[] years = {Calendar.getInstance().get(Calendar.YEAR) + "",
        (Calendar.getInstance().get(Calendar.YEAR) + 1) + ""};

JComboBox yearChoice = new JComboBox(years);
yearInformation.add((yearChoice));

JPanel dayMonthInformation = new JPanel();
dateInformation.add(dayMonthInformation);
dayMonthInformation.setLayout(new BoxLayout(dayMonthInformation, BoxLayout.X_AXIS));


dayMonthInformation.add(new JLabel(MONTH));
JComboBox monthSelect = new JComboBox(months);

monthSelect.setSelectedIndex(Calendar.getInstance().get(Calendar.MONTH));

dayMonthInformation.add(monthSelect);

int numberOfDaysInMonth = 31;

String[] days = new String[numberOfDaysInMonth];

for (int i = 0; i < numberOfDaysInMonth; i++) {
    days[i] = i + 1 + "";
}

JComboBox daySelect = new JComboBox(days);

dayMonthInformation.add(new JLabel(DAY));
dayMonthInformation.add(daySelect);

daySelect.setSelectedIndex(Calendar.getInstance().get(Calendar.DAY_OF_MONTH) - 1);

// make the date panel

JPanel symptomPanel = new JPanel();
totalPanel.add(symptomPanel);

symptomPanel.add(new JLabel(SYMPTOMS));
JTextArea symptomArea = new JTextArea(8, 40);
symptomPanel.add(symptomArea);

JPanel diagnosisPanel = new JPanel();
totalPanel.add(diagnosisPanel);

diagnosisPanel.add(new JLabel(DIAGNOSIS));
JTextArea diagnosisArea = new JTextArea(8, 40);
diagnosisPanel.add(diagnosisArea);

JPanel treatmentPanel = new JPanel();
totalPanel.add(treatmentPanel);

treatmentPanel.add(new JLabel(TREATMENT));
JTextArea treatmentArea = new JTextArea(8, 40);
treatmentPanel.add(treatmentArea);
```

```
        JPanel remarksPanel = new JPanel();
        totalPanel.add(remarksPanel);

        remarksPanel.add(new JLabel(REMARKS));
        JTextArea remarksArea = new JTextArea(8, 40);
        remarksPanel.add(remarksArea);

        JPanel bottomSection = new JPanel();
        totalPanel.add(bottomSection);

        bottomSection.setLayout(new BorderLayout());

        JButton saveButton = new JButton(SAVE);
        bottomSection.add(saveButton, BorderLayout.EAST);

        saveButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int date;
                String symptom;
                String diagnosis;
                String treatment;
                String remarks;

                String day = "" + daySelect.getSelectedItem();

                String month;

                if (monthSelect.getSelectedIndex() + 1 > 9)
                    month = "" + (monthSelect.getSelectedIndex() + 1);
                else
                    month = "0" + (monthSelect.getSelectedIndex() + 1);

                String year = "" + yearChoice.getSelectedItem();

                date = Integer.parseInt(day + month + year);

                symptom = symptomArea.getText();
                diagnosis = diagnosisArea.getText();
                treatment = treatmentArea.getText();
                remarks = remarksArea.getText();

                manager.findPatient(uniqueID).getRecord().enqueue((new Examination(date, symptom,
diagnosis, treatment, remarks)));

                addPatientExaminationFrame.dispose();
                editPatientFrame.dispose();
                createEditPatientDisplay(uniqueID);
                addPatientExaminationFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);

            }
        });

        JButton cancelButton = new JButton(CANCEL);
        bottomSection.add(cancelButton, BorderLayout.WEST);

        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addPatientExaminationFrame.dispose();
                addPatientExaminationFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        });

    }
```

```
    /**
     * Creates the frame used to edit the general information regarding a specified patient.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditPatientGeneralInfoDisplay(String uniqueID) {
        editPatientGeneralInfoFrame = new JFrame();
        editPatientGeneralInfoFrame.setAlwaysOnTop(true);

        createEditPatientGeneralInfoForm(uniqueID);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editPatientGeneralInfoFrame.dispose();
                editPatientGeneralInfoFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        };
        editPatientGeneralInfoFrame.addWindowListener(exitListener);
        editPatientGeneralInfoFrame
                .setSize(new Dimension(GUI.EDIT_PATIENT_GENERAL_FRAME_WIDTH,
GUI.EDIT_PATIENT_GENERAL_FRAME_HEIGHT));
        editPatientGeneralInfoFrame.setLocationRelativeTo(null);
        editPatientGeneralInfoFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display shown when editing the general information
     * regarding a specified patient.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditPatientGeneralInfoForm(String uniqueID) {

        JPanel totalPanel = new JPanel();
        totalPanel.setLayout(new BoxLayout(totalPanel, BoxLayout.Y_AXIS));

        editPatientGeneralInfoFrame.getContentPane().add(totalPanel);

        JLabel generalInfoLabel = new JLabel(GENERAL_INFORMATION);
        generalInfoLabel.setHorizontalAlignment(SwingConstants.CENTER);

        totalPanel.add(generalInfoLabel);

        JPanel generalPatientInformationEdit = new JPanel();
        totalPanel.add(generalPatientInformationEdit);
        generalPatientInformationEdit.setLayout(new GridLayout(0, 4));

        ArrayList<JTextField> textFields = new ArrayList<JTextField>();

        String[] labels1 = {NAME, SEX, SPECIES, BREED, COLOUR, BIRTHDATE,
                WEIGHT};

        Patient selectedPatient = manager.findPatient(uniqueID);

        String[] defaultInfo = {selectedPatient.getName(), selectedPatient.getSex() + "",
selectedPatient.getSpecies(),
                selectedPatient.getBreed(), selectedPatient.getColour(),
manager.changeDateFormat(selectedPatient.getDateOfBirth() + "", DATE_FORMAT_PLAIN,
DATE_FORMAT_DASH),
                selectedPatient.getWeight() + ""};

        JLabel label;

        // Creates the textfields for the patient's general information with them already filled
in.
        for (int i = 0; i < labels1.length; i++) {
```

```
            label = new JLabel(labels1[i]);
            label.setFont(new Font(label.getFont().getName(), Font.BOLD,
label.getFont().getSize()));
            generalPatientInformationEdit.add(label);
            textFields.add(new JTextField(defaultInfo[i]));
            generalPatientInformationEdit.add(textFields.get(i));
        }

        Border border = generalPatientInformationEdit.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        generalPatientInformationEdit.setBorder(new CompoundBorder(border, margin));

        JPanel bottomButtons = new JPanel();
        bottomButtons.setLayout(new BorderLayout());
        totalPanel.add(bottomButtons, BorderLayout.SOUTH);

        JButton saveButton = new JButton(SAVE);

        JButton cancelButton = new JButton(CANCEL);

        bottomButtons.add(cancelButton, BorderLayout.WEST);
        bottomButtons.add(saveButton, BorderLayout.EAST);

        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editPatientGeneralInfoFrame.dispose();
                editPatientGeneralInfoFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        });


        saveButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    for (int i = 0; i < textFields.size(); i++)
                        if (textFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();
                    String name = textFields.get(0).getText();
                    char sex = textFields.get(1).getText().toCharArray()[0];

                    String species = textFields.get(2).getText();
                    String breed = textFields.get(3).getText();
                    String colour = textFields.get(4).getText();
                    int dateOfBirth = Integer.parseInt(manager.changeDateFormat(
                            textFields.get(5).getText(), DATE_FORMAT_DASH, DATE_FORMAT_PLAIN
                    ));

                    double weight = Double.parseDouble(textFields.get(6).getText());

                    Patient patient = manager.findPatient(uniqueID);
                    patient.setName(name);
                    patient.setSex(sex);
                    patient.setSpecies(species);
                    patient.setBreed(breed);
                    patient.setColour(colour);
                    patient.setDateOfBirth(dateOfBirth);
                    patient.setWeight(weight);


                    editPatientGeneralInfoFrame.dispose();
                    editPatientFrame.dispose();
                    createEditPatientDisplay(uniqueID);
                    editPatientGeneralInfoFrame.setAlwaysOnTop(false);
                    editPatientFrame.setEnabled(true);
                    editPatientFrame.setAlwaysOnTop(true);
                } catch (ArrayIndexOutOfBoundsException e1) {
```

```
                    editPatientGeneralInfoFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editPatientGeneralInfoFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    editPatientGeneralInfoFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editPatientGeneralInfoFrame.setAlwaysOnTop(true);
                    }
                }

            }
        });
    }

    /**
     * Creates the frame to display the information of the specified patient's owner.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditOwnerInfoDisplay(String uniqueID) {
        editOwnerInfoFrame = new JFrame();
        editOwnerInfoFrame.setAlwaysOnTop(true);

        createEditOwnerInfoForm(uniqueID);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editOwnerInfoFrame.dispose();
                editOwnerInfoFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        };
        editOwnerInfoFrame.addWindowListener(exitListener);

        editOwnerInfoFrame.setSize(new Dimension(GUI.EDIT_OWNER_INFO_FRAME_WIDTH,
GUI.EDIT_OWNER_INFO_FRAME_HEIGHT));
        editOwnerInfoFrame.setLocationRelativeTo(null);
        editOwnerInfoFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display showing the information of the specified
     * patient's owner.
     *
     * @param uniqueID the unique ID of the specified patient
     */
    private void createEditOwnerInfoForm(String uniqueID) {
        JPanel ownerInformation = new JPanel();
        editOwnerInfoFrame.getContentPane().add(ownerInformation);

        ownerInformation.setLayout(new BoxLayout(ownerInformation, BoxLayout.Y_AXIS));
        JLabel ownerInformationLabel = new JLabel(OWNER_INFORMATION);
        ownerInformation.add(ownerInformationLabel);

        ownerInformationLabel.setHorizontalAlignment(SwingConstants.CENTER);

        JPanel generalOwnerInformationEdit = new JPanel();
        ownerInformation.add(generalOwnerInformationEdit);
        generalOwnerInformationEdit.setLayout(new GridLayout(0, 4));

        ArrayList<JTextField> textFields = new ArrayList<JTextField>();
```

```
        Patient selectedPatient = manager.findPatient(uniqueID);

        String[] labels2 = {FIRST_NAME, LAST_NAME, EMAIL, TELEPHONE_NUMBER, ADDRESS};

        String[] defaultInfo2 = {selectedPatient.getOwnerFirstName(),
selectedPatient.getOwnerLastName(),
                selectedPatient.getOwnerEmail(), selectedPatient.getOwnerNumber() + "",
                selectedPatient.getOwnerAddress()};

        JLabel label;

        // Displays textfields with the patient's owner's information already filled in.
        for (int i = 0; i < labels2.length; i++) {
            label = new JLabel(labels2[i]);
            label.setFont(new Font(label.getFont().getName(), Font.BOLD,
label.getFont().getSize()));
            generalOwnerInformationEdit.add(label);
            textFields.add(new JTextField(defaultInfo2[i]));
            generalOwnerInformationEdit.add(textFields.get(i));
        }

        Border border = generalOwnerInformationEdit.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        generalOwnerInformationEdit.setBorder(new CompoundBorder(border, margin));

        JPanel bottomButtons = new JPanel();
        bottomButtons.setLayout(new BorderLayout());
        ownerInformation.add(bottomButtons);

        JButton saveButton = new JButton(SAVE);
        JButton cancelButton = new JButton(CANCEL);

        bottomButtons.add(cancelButton, BorderLayout.WEST);
        bottomButtons.add(saveButton, BorderLayout.EAST);

        cancelButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editOwnerInfoFrame.dispose();
                editOwnerInfoFrame.setAlwaysOnTop(false);
                editPatientFrame.setEnabled(true);
                editPatientFrame.setAlwaysOnTop(true);
            }
        });

        saveButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    for (int i = 0; i < textFields.size(); i++)
                        if (textFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();
                    String ownerFirstName = textFields.get(0).getText();
                    String ownerLastName = textFields.get(1).getText();
                    String ownerEmail = textFields.get(2).getText();
                    if(!ownerEmail.contains("@"))
                        throw new NumberFormatException();
                    long ownerNumber = Long.parseLong(textFields.get(3).getText());


                    String ownerAddress = textFields.get(4).getText();

                    Patient patient = manager.findPatient(uniqueID);

                    patient.setOwnerFirstName(ownerFirstName);
                    patient.setOwnerLastName(ownerLastName);
                    patient.setOwnerEmail(ownerEmail);
                    patient.setOwnerNumber(ownerNumber);
                    patient.setOwnerAddress(ownerAddress);
```

```
                    editOwnerInfoFrame.dispose();
                    editOwnerInfoFrame.setAlwaysOnTop(false);
                    editPatientFrame.dispose();
                    createEditPatientDisplay(uniqueID);
                    editPatientFrame.setEnabled(true);
                    editPatientFrame.setAlwaysOnTop(true);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editOwnerInfoFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editOwnerInfoFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    editOwnerInfoFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editOwnerInfoFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

    }

    /**
     * Creates the frame to edit the general information about the specified doctor.
     *
     * @param uniqueID the unique ID of the specified doctor
     */
    private void createEditDoctorGeneralDisplay(String uniqueID) {
        editDoctorGeneralInfoFrame = new JFrame();
        editDoctorGeneralInfoFrame.setAlwaysOnTop(true);
        createEditDoctorGeneralForm(uniqueID);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editDoctorGeneralInfoFrame.dispose();
                editDoctorGeneralInfoFrame.setAlwaysOnTop(false);
                editDoctorFrame.setEnabled(true);
                editDoctorFrame.setAlwaysOnTop(true);
            }
        };
        editDoctorGeneralInfoFrame.addWindowListener(exitListener);

        editDoctorGeneralInfoFrame
                .setSize(new Dimension(GUI.EDIT_DOCTOR_GENERAL_FRAME_WIDTH,
GUI.EDIT_DOCTOR_GENERAL_FRAME_HEIGHT));
        editDoctorGeneralInfoFrame.setLocationRelativeTo(null);
        editDoctorGeneralInfoFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display shown when editing the general information
     * about the specified doctor.
     *
     * @param uniqueID the unique ID of the specified doctor
     */
    private void createEditDoctorGeneralForm(String uniqueID) {
        JPanel totalEdit = new JPanel();
        editDoctorGeneralInfoFrame.getContentPane().add(totalEdit);

        totalEdit.setLayout(new BoxLayout(totalEdit, BoxLayout.Y_AXIS));

        JPanel generalInfo = new JPanel();
        generalInfo.setLayout(new GridLayout(0, 4));
```

```
        totalEdit.add(generalInfo);

        String[] labels = {FIRST_NAME, LAST_NAME, EMAIL, TELEPHONE_NUMBER, ADDRESS};

        Doctor doctor = manager.findDoctor(uniqueID);

        String[] defaultInfo = {doctor.getFirstName(), doctor.getLastName(), doctor.getEmail(),
                doctor.getNumber() + "", doctor.getAddress()};

        JLabel label;

        ArrayList<JTextField> infoFields = new ArrayList<JTextField>();

        // Displays all the textfields containing the general information about the doctor
already filled in.
        for (int i = 0; i < defaultInfo.length; i++) {
            label = new JLabel(labels[i]);
            label.setFont(new Font(label.getFont().getName(), Font.BOLD,
label.getFont().getSize()));
            generalInfo.add(label);
            infoFields.add(new JTextField(defaultInfo[i]));
            generalInfo.add(infoFields.get(i));
        }

        Border border = generalInfo.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        generalInfo.setBorder(new CompoundBorder(border, margin));

        JPanel bottomSection = new JPanel();
        totalEdit.add(bottomSection);

        bottomSection.setLayout(new BorderLayout());

        JButton saveButton = new JButton(SAVE);
        bottomSection.add(saveButton, BorderLayout.EAST);

        saveButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    for (int i = 0; i < infoFields.size(); i++)
                        if (infoFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();

                    Doctor doctor = manager.findDoctor(uniqueID);
                    doctor.setFirstName(infoFields.get(0).getText());
                    doctor.setLastName(infoFields.get(1).getText());
                    doctor.setEmail(infoFields.get(2).getText());
                    doctor.setNumber(Integer.parseInt(infoFields.get(3).getText()));
                    doctor.setAddress(infoFields.get(4).getText());

                    editDoctorGeneralInfoFrame.dispose();
                    editDoctorFrame.dispose();
                    createEditDoctorDisplay(uniqueID);
                    editDoctorGeneralInfoFrame.setAlwaysOnTop(false);
                    editDoctorFrame.setEnabled(true);
                    editDoctorFrame.setAlwaysOnTop(true);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editDoctorGeneralInfoFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editDoctorGeneralInfoFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    editDoctorGeneralInfoFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editDoctorGeneralInfoFrame.setAlwaysOnTop(true);
```

```
                    }
                }
            }
        });

        JButton cancelButton = new JButton(CANCEL);
        bottomSection.add(cancelButton, BorderLayout.WEST);

        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                editDoctorGeneralInfoFrame.dispose();
                editDoctorGeneralInfoFrame.setAlwaysOnTop(false);
                editDoctorFrame.setEnabled(true);
                editDoctorFrame.setAlwaysOnTop(true);
            }
        });


    }

    /**
     * Creates the frame used to edit information about the specified doctor.
     *
     * @param uniqueID the unique ID of the specified doctor
     */
    private void createEditDoctorDisplay(String uniqueID) {
        mainFrame.setEnabled(false);
        mainFrame.setAlwaysOnTop(false);
        editDoctorFrame = new JFrame();
        editDoctorFrame.setAlwaysOnTop(true);

        createEditDoctorForm(uniqueID);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editDoctorFrame.dispose();
                editDoctorFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
            }
        };

        editDoctorFrame.addWindowListener(exitListener);

        editDoctorFrame.setSize(new Dimension(GUI.EDIT_DOCTOR_FRAME_WIDTH,
GUI.EDIT_DOCTOR_FRAME_HEIGHT));
        editDoctorFrame.setLocationRelativeTo(null);
        editDoctorFrame.setVisible(true);
    }


    /**
     * Creates the contents of the display shown when editing information about
     * the specified doctor.
     *
     * @param uniqueID the unique ID of the specified doctor
     */
    private void createEditDoctorForm(String uniqueID) {

        JPanel totalEdit = new JPanel();
        editDoctorFrame.getContentPane().add(totalEdit);

        totalEdit.setLayout(new BoxLayout(totalEdit, BoxLayout.Y_AXIS));



        JPanel generalInfoLabelPanel = new JPanel();
        JButton removeDoctorButton = new JButton("Remove Doctor");
        generalInfoLabelPanel.add(removeDoctorButton);
```

```
        removeDoctorButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editDoctorFrame.setAlwaysOnTop(false);
                int result = JOptionPane.showConfirmDialog(null, "Are you sure that you would
like to remove this doctor from the system?", ERROR, JOptionPane.YES_NO_OPTION);
                if (result == JOptionPane.YES_OPTION) {
                    editDoctorFrame.setAlwaysOnTop(true);
                    for(int i = 0; i < manager.getReceptionist().getAppointments().size();i++)
                    {

if(manager.getReceptionist().getAppointments().get(i).getDoctor().getUniqueID()
                                .equals(uniqueID))
                            manager.getReceptionist().getAppointments().remove(i);
                    }
                    manager.findDoctor(uniqueID).getTasks().clear();
                    manager.getDoctors().remove(manager.findDoctor(uniqueID));

                    editDoctorFrame.dispose();
                    searchDoctor.doClick();
                    mainFrame.setEnabled(true);
                    mainFrame.setAlwaysOnTop(true);
                }
                else if(result == JOptionPane.NO_OPTION)
                {
                    editDoctorFrame.setAlwaysOnTop(true);
                }

            }

        });

        generalInfoLabelPanel.add(new JLabel(GENERAL_INFORMATION));



        JButton editGeneralInfoButton = new JButton(EDIT);

        editGeneralInfoButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createEditDoctorGeneralDisplay(uniqueID);
                editDoctorFrame.setEnabled(false);
                editDoctorFrame.setAlwaysOnTop(false);
            }
        });

        generalInfoLabelPanel.add(editGeneralInfoButton);

        totalEdit.add(generalInfoLabelPanel);

        JPanel generalInfo = new JPanel();
        generalInfo.setLayout(new GridLayout(0, 4));

        totalEdit.add(generalInfo);

        String[] labels = {FIRST_NAME, LAST_NAME, EMAIL, TELEPHONE_NUMBER, ADDRESS};

        Doctor doctor = manager.findDoctor(uniqueID);

        String[] defaultInfo = {doctor.getFirstName(), doctor.getLastName(), doctor.getEmail(),
                doctor.getNumber() + "", doctor.getAddress()};

        JLabel label;

        ArrayList<JLabel> infoLabels = new ArrayList<JLabel>();

        // Displays general information about the doctor.
        for (int i = 0; i < defaultInfo.length; i++) {
            label = new JLabel(labels[i]);
```

```
            label.setFont(new Font(label.getFont().getName(), Font.BOLD,
label.getFont().getSize()));
            generalInfo.add(label);
            infoLabels.add(new JLabel(defaultInfo[i]));
            generalInfo.add(infoLabels.get(i));
        }

        Border border = generalInfo.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        generalInfo.setBorder(new CompoundBorder(border, margin));

        JPanel taskPanel = new JPanel();


        taskPanel.add(new JLabel(TASKS));

        JButton addTaskButton = new JButton(ADD);
        taskPanel.add(addTaskButton);

        addTaskButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                createAddTaskDisplay(uniqueID);
                editDoctorFrame.setEnabled(false);
                editDoctorFrame.setAlwaysOnTop(false);
            }
        });

        JButton removeTaskButton = new JButton(REMOVE);
        taskPanel.add(removeTaskButton);

        totalEdit.add(taskPanel);

        JPanel taskBoxes = new JPanel();

        taskBoxes.setLayout(new BoxLayout(taskBoxes, BoxLayout.Y_AXIS));

        ArrayList<JButton> taskBoxPanels = new ArrayList<>();

        ArrayList<Task> tasks = manager.findDoctor(uniqueID).getTasks();

        String[][] data = new String[manager.findDoctor(uniqueID).getTasks().size()][3];

        for (int i = 0; i < manager.findDoctor(uniqueID).getTasks().size(); i++) {
            data[i][0] = tasks.get(i).getPurpose();
            data[i][1] = manager.changeDateFormat(tasks.get(i).getDate() + "", DATE_FORMAT_PLAIN,
DATE_FORMAT_DASH);
            data[i][2] = manager.returnFormattedTime(tasks.get(i).getStartTime(),
tasks.get(i).getEndTime());
        }

        JTable taskTable = new JTable(data, appointmentColumns);
        JScrollPane pane = new JScrollPane(taskTable);
        taskTable.setDefaultEditor(Object.class, null);

        JPanel taskTablePanel = new JPanel();
        taskTablePanel.setLayout(new BorderLayout());
        taskTablePanel.add(pane, BorderLayout.CENTER);
        totalEdit.add(taskTablePanel);

        removeTaskButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    int selectedRow = taskTable.getSelectedRow();
                    Doctor doctor = manager.findDoctor(uniqueID);


                    if (doctor.getTasks().get(selectedRow).getPurpose().contains(APPOINTMENT))
                        for (int i = 0; i < manager.getReceptionist().getAppointments().size();
i++)
```

```
                            if (manager.getReceptionist().getAppointments().get(i).getUniqueID().
                                    equals(doctor.getTasks().get(selectedRow).getUniqueID()))
                                manager.getReceptionist().getAppointments().remove(i);


                        doctor.getTasks().remove(selectedRow);
                        editDoctorFrame.dispose();
                        createEditDoctorDisplay(uniqueID);
                    } catch (ArrayIndexOutOfBoundsException e1) {
                        editDoctorFrame.setAlwaysOnTop(false);
                        int result = JOptionPane.showConfirmDialog(null, SELECT_TASK_FIRST, ERROR,
JOptionPane.DEFAULT_OPTION);
                        if (result == JOptionPane.OK_OPTION) {
                            editDoctorFrame.setAlwaysOnTop(true);
                        }
                    }

                }
            });


        }


        /**
         * Creates the frame used to add a task to the specified doctor.
         *
         * @param uniqueID the unique ID of the specified doctor
         */
        private void createAddTaskDisplay(String uniqueID){
            addTaskFrame = new JFrame();
            addTaskFrame.setAlwaysOnTop(true);

            createAddTaskForm(uniqueID);

            WindowListener exitListener = new WindowAdapter() {
                @Override
                public void windowClosing(WindowEvent e) {
                    addTaskFrame.dispose();
                    addTaskFrame.setAlwaysOnTop(false);
                    editDoctorFrame.setEnabled(true);
                    editDoctorFrame.setAlwaysOnTop(true);
                }
            };
            addTaskFrame.addWindowListener(exitListener);
            addTaskFrame.setSize(new Dimension(GUI.EDIT_DOCTOR_FRAME_WIDTH + 100,
GUI.EDIT_DOCTOR_FRAME_HEIGHT + 100));
            addTaskFrame.setLocationRelativeTo(null);
            addTaskFrame.setVisible(true);

        }

        /**
         * Creates the contents of the display shown when adding a task to the
         * specified doctor.
         *
         * @param uniqueID the unique ID of the specified doctor
         */
        private void createAddTaskForm(String uniqueID) {

            addTaskFrame = new JFrame();
            addTaskFrame.setAlwaysOnTop(true);
            addTaskFrame.setLayout(new BorderLayout());

            JPanel taskInformation = new JPanel();
            addTaskFrame.add(taskInformation, BorderLayout.NORTH);

            JLabel taskInfoLabel = new JLabel(TASK_INFORMATION);
            taskInformation.add(taskInfoLabel);
```

```
JPanel topSection = new JPanel();
JPanel shortInformation = new JPanel();
addTaskFrame.add(topSection, BorderLayout.CENTER);
topSection.setLayout(new BoxLayout(topSection, BoxLayout.PAGE_AXIS));
topSection.add(shortInformation);

Border border = shortInformation.getBorder();
Border margin = new EmptyBorder(20, 20, 20, 20);
shortInformation.setBorder(new CompoundBorder(border, margin));

shortInformation.setLayout(new BoxLayout(shortInformation, BoxLayout.Y_AXIS));
JPanel mainPurpose = new JPanel();
mainPurpose.add(new JLabel(MAIN_PURPOSE));

JTextField mainPurposeField = new JTextField(15);
mainPurpose.add(mainPurposeField);

shortInformation.add(mainPurpose);

JPanel description = new JPanel();
topSection.add(description);
description.setLayout(new BoxLayout(description, BoxLayout.X_AXIS));
description.setBorder(new CompoundBorder(description.getBorder(), margin));
description.add(new JLabel(TASK_DESCRIPTION));

JTextArea descriptionArea = new JTextArea(10, 10);
description.add(new JScrollPane(descriptionArea), BorderLayout.PAGE_START);

JPanel totalBottomSection = new JPanel();
JPanel bottomSection = new JPanel();
JPanel bottomButtons = new JPanel();
addTaskFrame.add(totalBottomSection, BorderLayout.SOUTH);
totalBottomSection.setLayout(new BoxLayout(totalBottomSection, BoxLayout.Y_AXIS));
totalBottomSection.add(bottomSection);
totalBottomSection.add(bottomButtons);
bottomSection.setLayout(new BoxLayout(bottomSection, BoxLayout.X_AXIS));

JPanel dateInformation = new JPanel();
bottomSection.add(dateInformation);
dateInformation.setLayout(new BoxLayout(dateInformation, BoxLayout.Y_AXIS));

JPanel yearInformation = new JPanel();
dateInformation.add(yearInformation);

yearInformation.setLayout(new BoxLayout(yearInformation, BoxLayout.X_AXIS));
yearInformation.add(new JLabel(YEAR));

String[] years = {Calendar.getInstance().get(Calendar.YEAR) + "",
        (Calendar.getInstance().get(Calendar.YEAR) + 1) + ""};

JComboBox yearChoice = new JComboBox(years);
yearInformation.add((yearChoice));

JPanel dayMonthInformation = new JPanel();
dateInformation.add(dayMonthInformation);
dayMonthInformation.setLayout(new BoxLayout(dayMonthInformation, BoxLayout.X_AXIS));

dayMonthInformation.add(new JLabel(MONTH));
JComboBox monthSelect = new JComboBox(months);

monthSelect.setSelectedIndex(Calendar.getInstance().get(Calendar.MONTH));

dayMonthInformation.add(monthSelect);

int numberOfDaysInMonth = 31;

String[] days = new String[numberOfDaysInMonth];
```

```
for (int i = 0; i < numberOfDaysInMonth; i++) {
    days[i] = i + 1 + "";
}

JComboBox daySelect = new JComboBox(days);

dayMonthInformation.add(new JLabel(DAY));
dayMonthInformation.add(daySelect);

daySelect.setSelectedIndex(Calendar.getInstance().get(Calendar.DAY_OF_MONTH) - 1);

JPanel timeInformation = new JPanel();
bottomSection.add(timeInformation);

timeInformation.setLayout(new GridLayout(0, 4));
timeInformation.add(new JLabel(START_TIME));

String[] hours = new String[12];
for (int i = 0; i < 12; i++) {
    hours[i] = i + 1 + "";
}

JComboBox startHour = new JComboBox(hours);
timeInformation.add(startHour);

JComboBox startMinute = new JComboBox(minutes);
timeInformation.add(startMinute);


JComboBox startDayOrNight = new JComboBox(dayOrNight);
timeInformation.add(startDayOrNight);

timeInformation.add(new JLabel(END_TIME));

JComboBox endHour = new JComboBox(hours);
timeInformation.add(endHour);

JComboBox endMinute = new JComboBox(minutes);
timeInformation.add(endMinute);

JComboBox endDayOrNight = new JComboBox(dayOrNight);
timeInformation.add(endDayOrNight);

// Sets the date and time in the combobox to the current date and time.
monthSelect.setSelectedIndex(Calendar.getInstance().get(Calendar.MONTH));
daySelect.setSelectedIndex(Calendar.getInstance().get(Calendar.DAY_OF_MONTH) - 1);
startHour.setSelectedIndex(Calendar.getInstance().get(Calendar.HOUR) - 1);
endHour.setSelectedIndex(Calendar.getInstance().get(Calendar.HOUR) - 1);
startMinute.setSelectedIndex(Calendar.getInstance().get(Calendar.MINUTE) / 15 - 1);
endMinute.setSelectedIndex(Calendar.getInstance().get(Calendar.MINUTE) / 15 - 1);
startDayOrNight.setSelectedIndex(Calendar.getInstance().get(Calendar.AM_PM));
endDayOrNight.setSelectedIndex(Calendar.getInstance().get(Calendar.AM_PM));

bottomButtons.setLayout(new BoxLayout(bottomButtons, BoxLayout.X_AXIS));

JButton cancelButton = new JButton(CANCEL);
JButton createTaskButton = new JButton(CREATE_TASK);

bottomButtons.add(cancelButton);
bottomButtons.add(createTaskButton);

cancelButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        addTaskFrame.dispose();
        addTaskFrame.setAlwaysOnTop(false);
        editDoctorFrame.setEnabled(true);
        editDoctorFrame.setAlwaysOnTop(true);
    }
});
```

```
        createTaskButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    if(mainPurposeField.getText().trim().equals("") ||
                            descriptionArea.getText().trim().equals(""))
                        throw new ArrayIndexOutOfBoundsException();
                    String purpose = mainPurposeField.getText();
                    String description = descriptionArea.getText();
                    Doctor doctor = manager.findDoctor(uniqueID);

                    String day = "" + daySelect.getSelectedIndex();

                    String month;

                    if (monthSelect.getSelectedIndex() > 8)
                        month = "" + (monthSelect.getSelectedIndex() + 1);
                    else
                        month = "0" + (monthSelect.getSelectedIndex() + 1);

                    String year = "" + yearChoice.getSelectedItem();

                    int date = Integer.parseInt(day + month + year);

                    int startTime = Integer.parseInt((String) startHour.getSelectedItem() +
startMinute.getSelectedItem())
                            * manager.amOrPm(startDayOrNight.getSelectedIndex());

                    int endTime = Integer.parseInt((String) endHour.getSelectedItem() +
endMinute.getSelectedItem())
                            * manager.amOrPm(endDayOrNight.getSelectedIndex());

                    manager.findDoctor(uniqueID).getTasks()
                            .add(new Task(purpose, description, doctor, date, startTime,
endTime));

                    addTaskFrame.dispose();
                    addTaskFrame.setAlwaysOnTop(false);
                    editDoctorFrame.dispose();
                    createEditDoctorDisplay(uniqueID);
                    editDoctorFrame.setEnabled(true);
                    editDoctorFrame.setAlwaysOnTop(true);
                }
                catch(ArrayIndexOutOfBoundsException | NumberFormatException e1)
                {
                    addTaskFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addTaskFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

        totalBottomSection.setBorder(new CompoundBorder(totalBottomSection.getBorder(), new
EmptyBorder(0, 20, 0, 20)));
    }

    /**
     * Creates the buttons used to display the different patients in the system.
     *
     * @param sortedPatients the list of patient to be displayed
     */
    private void createPatientsDisplayFeed(ArrayList<Patient> sortedPatients) {
        firstTabContents.setPreferredSize(new Dimension(mainFrame.getSize().width,
                (int) (mainFrame.getSize().height - mainFrame.getSize().height * 0.12)));
        firstTabContents.add(patientBoxes, BorderLayout.CENTER);
        patientBoxes.setLayout(new BoxLayout(patientBoxes, BoxLayout.PAGE_AXIS));
```

```
        Collections.sort(sortedPatients, new Comparator<Patient>() {
            public int compare(Patient p1, Patient p2) {
                if (p1.getName().equals(p2.getName()))
                    return 0;

                char[] p1Char = p1.getName().toCharArray();
                char[] p2Char = p2.getName().toCharArray();

                int length = p1Char.length;
                if (p1Char.length > p2Char.length) {
                    length = p2Char.length;
                }

                for (int i = 0; i < length; i++) {
                    if (p1Char[i] < p2Char[i])
                        return -1;
                    else if (p1Char[i] > p2Char[i])
                        return 1;
                }

                return length == p1Char.length ? -1 : 1;
            }
        });

        for (int i = 0; i < sortedPatients.size(); i++) {
            patientBoxPanels.add(new JButton());
            patientBoxes.add(patientBoxPanels.get(i));
            patientBoxPanels.get(i).setName(i + "");

            patientBoxPanels.get(i).setLayout(new GridLayout(0, 3));
            patientBoxPanels.get(i).setPreferredSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 30,
50));
            patientBoxPanels.get(i).setMaximumSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 30, 50));

            patientBoxPanels.get(i).add(new JLabel(OWNER +
sortedPatients.get(i).getOwnerFirstName() + " "
                    + sortedPatients.get(i).getOwnerLastName()), BorderLayout.WEST);
            patientBoxPanels.get(i).add(new JLabel(PATIENT + ": " +
sortedPatients.get(i).getName()), BorderLayout.CENTER);
            patientBoxPanels.get(i).add(new JLabel(SPECIES + sortedPatients.get(i).getSpecies()),
                    BorderLayout.EAST);

            // Opens the edit display of the patient whose button is clicked.
            patientBoxPanels.get(i).addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    Object source = e.getSource();
                    int selectedPanel = -1;
                    for (int a = 0; a < sortedPatients.size() && selectedPanel == -1; a++)
                        if (((JButton)
source).getName().equals(patientBoxPanels.get(a).getName())) {
                            selectedPanel = a;
                        }

                    createEditPatientDisplay(sortedPatients.get(selectedPanel).getUniqueID());
                }
            });
        }

        pane = new JScrollPane(patientBoxes);
        firstTabContents.add(pane);
        pane.setViewportView(patientBoxes);
        pane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    }

    /**
     * Creates the buttons used to display the different doctors in the system.
     *
     * @param sortedDoctors the list of doctors to be displayed
     */
```

```
    private void createDoctorsDisplayFeed(ArrayList<Doctor> sortedDoctors) {
        firstTabContents.setPreferredSize(new Dimension(mainFrame.getSize().width,
                (int) (mainFrame.getSize().height - mainFrame.getSize().height * 0.12)));
        firstTabContents.add(doctorBoxes, BorderLayout.CENTER);
        doctorBoxes.setLayout(new BoxLayout(doctorBoxes, BoxLayout.PAGE_AXIS));

        // Sorts the doctors based on their first and last name.
        Collections.sort(sortedDoctors, new Comparator<Doctor>() {
            public int compare(Doctor d1, Doctor d2) {
                if (d1.getFirstName().equals(d2.getFirstName()))
                    if (d1.getLastName().equals(d2.getLastName()))
                        return 0;

                if (d1.getFirstName().equals(d2.getFirstName())) {
                    char[] p1Char = d1.getLastName().toCharArray();
                    char[] p2Char = d2.getLastName().toCharArray();

                    int length = p1Char.length;
                    if (p1Char.length > p2Char.length) {
                        length = p2Char.length;
                    }

                    for (int i = 0; i < length; i++) {
                        if (p1Char[i] < p2Char[i])
                            return -1;
                        else if (p1Char[i] > p2Char[i])
                            return 1;
                    }

                    return length == p1Char.length ? -1 : 1;
                } else {
                    char[] p1Char = d1.getFirstName().toCharArray();
                    char[] p2Char = d2.getFirstName().toCharArray();

                    int length = p1Char.length;
                    if (p1Char.length > p2Char.length) {
                        length = p2Char.length;
                    }

                    for (int i = 0; i < length; i++) {
                        if (p1Char[i] < p2Char[i])
                            return -1;
                        else if (p1Char[i] > p2Char[i])
                            return 1;
                    }

                    return length == p1Char.length ? -1 : 1;
                }
            }
        });

        for (int i = 0; i < sortedDoctors.size(); i++) {
            doctorBoxPanels.add(new JButton());
            doctorBoxes.add(doctorBoxPanels.get(i));
            doctorBoxPanels.get(i).setName(i + "");

            doctorBoxPanels.get(i).setLayout(new GridLayout(0, 3));
            doctorBoxPanels.get(i).setPreferredSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 30,
50));
            doctorBoxPanels.get(i).setMaximumSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 30, 50));

            doctorBoxPanels.get(i).add(new JLabel(FIRST_NAME +
sortedDoctors.get(i).getFirstName()),
                    BorderLayout.WEST);
            doctorBoxPanels.get(i).add(new JLabel(LAST_NAME +
sortedDoctors.get(i).getLastName()),
                    BorderLayout.CENTER);
            doctorBoxPanels.get(i).add(new JLabel(EMAIL + sortedDoctors.get(i).getEmail()),
BorderLayout.EAST);
```

```
            // Opens the edit display of the doctor whose button is clicked.
            doctorBoxPanels.get(i).addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    Object source = e.getSource();
                    int selectedPanel = -1;
                    for (int a = 0; a < sortedDoctors.size() && selectedPanel == -1; a++)
                        if (((JButton)
                                source).getName
                                ().equals(doctorBoxPanels
                                .get(a).getName())) {
                            selectedPanel = a;
                        }

                    createEditDoctorDisplay(sortedDoctors.get(selectedPanel).getUniqueID());
                }


            });
        }

        pane = new JScrollPane(doctorBoxes);
        firstTabContents.add(pane);
        pane.setViewportView(doctorBoxes);
        pane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    }

    /**
     * Creates the options allowing the user to sort appointments based on date, doctor and
patient.
     * It also allowed the user to add a new appointment. These options are only visible if the
third tab
     * is selected.
     */
    private void createTopOptionsApp() {
        JPanel bottomOptions = new JPanel();
        bottomOptions.setLayout(new FlowLayout());

        secondTabContents.add(bottomOptions, BorderLayout.PAGE_START);

        JLabel sortLabel = new JLabel(SORT_BY);
        bottomOptions.add(sortLabel);

        JButton dateButton = new JButton(DATE.substring(0,DATE.length()-2));
        bottomOptions.add(dateButton);

        dateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                appointmentBoxes = new JPanel();
                appointmentBoxPanels = new ArrayList<JButton>();
                secondTabContents.removeAll();
                createTopOptionsApp();
                createAppointmentsDisplayFeed(SORT_BY_DATE);
                secondTabContents.revalidate();
            }
        });

        doctorButton = new JButton(DOCTOR);
        bottomOptions.add(doctorButton);

        doctorButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                appointmentBoxes = new JPanel();
                appointmentBoxPanels = new ArrayList<JButton>();
                secondTabContents.removeAll();
                createTopOptionsApp();
                createAppointmentsDisplayFeed(SORT_BY_DOCTORS);
```

```
                secondTabContents.revalidate();
            }
        });

        JButton patientButton = new JButton(PATIENT);
        bottomOptions.add(patientButton);


        patientButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                appointmentBoxes = new JPanel();
                appointmentBoxPanels = new ArrayList<JButton>();
                secondTabContents.removeAll();
                createTopOptionsApp();
                createAppointmentsDisplayFeed(SORT_BY_PATIENTS);
                secondTabContents.revalidate();
            }
        });

        JButton newAppointment = new JButton(NEW_APPOINTMENT);
        bottomOptions.add(newAppointment);

        newAppointment.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                mainFrame.setEnabled(false);
                mainFrame.setAlwaysOnTop(false);
                createAddAppointmentDisplay(null);
            }
        });
    }

    /**
     * Creates the frame that is used to add appointments. If the appointment is added from a
     * specific patient's edit frame, then the combobox for the patient is already selected
     * and disabled. If the appointment is added from the second tab, then the patient can be
     * selected and the unique ID will be null.
     *
     * @param uniqueID the unique ID of the patient from whose edit frame, the new appointment
button was pressed.
     */
    private void createAddAppointmentDisplay(String uniqueID) {
        addAppointmentFrame = new JFrame();
        addAppointmentFrame.setLayout(new BorderLayout());
        addAppointmentFrame.setAlwaysOnTop(true);
        createAddAppointmentForm(uniqueID);

        addAppointmentFrame
                .setSize(new Dimension(GUI.ADD_APPOINTMENT_FRAME_WIDTH,
GUI.ADD_APPOINTMENT_FRAME_HEIGHT + 250));
        addAppointmentFrame.setLocationRelativeTo(null);
        addAppointmentFrame.setVisible(true);
    }

    /**
     * Creates the contents for the second tab of the display allowing the user to create a
     * new appointment.
     *
     * @param uniqueID the unique ID of the patient, if this display was opened from their edit
frame
     */
    private void createAddAppointmentForm(String uniqueID) {
        JPanel appointmentInformation = new JPanel();
        addAppointmentFrame.add(appointmentInformation, BorderLayout.NORTH);

        JLabel taskInfoLabel = new JLabel(APPOINTMENT_INFORMATION);
        appointmentInformation.add(taskInfoLabel);

        JPanel topSection = new JPanel();
```

```
        JPanel shortInformation = new JPanel();
        addAppointmentFrame.add(topSection, BorderLayout.CENTER);
        topSection.setLayout(new BoxLayout(topSection, BoxLayout.PAGE_AXIS));
        topSection.add(shortInformation);

        Border border = shortInformation.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        shortInformation.setBorder(new CompoundBorder(border, margin));

        shortInformation.setLayout(new GridLayout(0, 4));
        shortInformation.add(new JLabel(APPOINTMENT_TYPE));
        JTextField appointmentTypeField = new JTextField(15);
        shortInformation.add(appointmentTypeField);
        shortInformation.add(new JLabel(MAIN_PURPOSE));
        JTextField mainPurposeField = new JTextField(15);
        shortInformation.add(mainPurposeField);
        shortInformation.add(new JLabel(DOCTOR));

        // Allows the user to select the doctor for this appointment.
        String[] doctors = new String[manager.getDoctors().size() + 1];
        doctors[0] = ASSIGN_DOCTOR_AUTOMATICALLY;
        for (int i = 0; i < manager.getDoctors().size(); i++) {
            doctors[i + 1] = manager.getDoctors().get(i).getFirstName() + " " +
manager.getDoctors().get(i).getLastName();
        }
        JComboBox doctorOptions = new JComboBox(doctors);
        shortInformation.add(doctorOptions);

        shortInformation.add(new JLabel(PATIENT));

        // Allows the user to select the patient for this appointment.
        String[] patients = new String[manager.getPatients().size()];
        for (int i = 0; i < manager.getPatients().size(); i++) {
            patients[i] = manager.getPatients().get(i).getName() + " (" + OWNER
                    + manager.getPatients().get(i).getOwnerFirstName() + " "
                    + manager.getPatients().get(i).getOwnerLastName() + ")";
        }
        JComboBox patientOptions = new JComboBox(patients);
        shortInformation.add(patientOptions);

        // If this frame was opened from a patient's edit display, it disables
        // the selection of a patient.
        if (uniqueID != null) {
            Patient patient;
            int index = 0;
            for (int i = 0; i < manager.getPatients().size(); i++) {
                patient = manager.getPatients().get(i);

                if (patient.getUniqueID().equals(uniqueID)) {
                    index = i;
                }
            }
            patientOptions.setSelectedIndex(index);
            patientOptions.setEditable(false);
            patientOptions.setEnabled(false);

        }

        JPanel description = new JPanel();
        topSection.add(description);
        description.setLayout(new BoxLayout(description, BoxLayout.X_AXIS));
        description.setBorder(new CompoundBorder(description.getBorder(), margin));
        description.add(new JLabel(APPOINTMENT_DESCRIPTION));
        JTextArea descriptionArea = new JTextArea(10, 10);
        description.add(new JScrollPane(descriptionArea), BorderLayout.PAGE_START);

        JPanel totalBottomSection = new JPanel();
        JPanel bottomSection = new JPanel();
        JPanel bottomButtons = new JPanel();
```

```
addAppointmentFrame.add(totalBottomSection, BorderLayout.SOUTH);
totalBottomSection.setLayout(new BoxLayout(totalBottomSection, BoxLayout.Y_AXIS));
totalBottomSection.add(bottomSection);
totalBottomSection.add(bottomButtons);
bottomSection.setLayout(new BoxLayout(bottomSection, BoxLayout.X_AXIS));

JPanel dateInformation = new JPanel();
bottomSection.add(dateInformation);
dateInformation.setLayout(new BoxLayout(dateInformation, BoxLayout.Y_AXIS));

JPanel yearInformation = new JPanel();
dateInformation.add(yearInformation);

yearInformation.setLayout(new BoxLayout(yearInformation, BoxLayout.X_AXIS));
yearInformation.add(new JLabel(YEAR));

String[] years = {Calendar.getInstance().get(Calendar.YEAR) + "",
        (Calendar.getInstance().get(Calendar.YEAR) + 1) + ""};

JComboBox yearChoice = new JComboBox(years);
yearInformation.add((yearChoice));

JPanel dayMonthInformation = new JPanel();
dateInformation.add(dayMonthInformation);
dayMonthInformation.setLayout(new BoxLayout(dayMonthInformation, BoxLayout.X_AXIS));

dayMonthInformation.add(new JLabel(MONTH));
JComboBox monthSelect = new JComboBox(months);

dayMonthInformation.add(monthSelect);

int numberOfDaysInMonth = 31;

String[] days = new String[numberOfDaysInMonth];

for (int i = 0; i < numberOfDaysInMonth; i++) {
    days[i] = i + 1 + "";
}

dayMonthInformation.add(new JLabel(DAY));

JComboBox daySelect = new JComboBox(days);
dayMonthInformation.add(daySelect);

JPanel timeInformation = new JPanel();
bottomSection.add(timeInformation);

timeInformation.setLayout(new GridLayout(0, 4));
timeInformation.add(new JLabel(START_TIME));

String[] hours = new String[12];
for (int i = 0; i < 12; i++) {
    hours[i] = i + 1 + "";
}

JComboBox startHour = new JComboBox(hours);
timeInformation.add(startHour);

JComboBox startMinute = new JComboBox(minutes);
timeInformation.add(startMinute);


JComboBox startDayOrNight = new JComboBox(dayOrNight);
timeInformation.add(startDayOrNight);

timeInformation.add(new JLabel(END_TIME));

JComboBox endHour = new JComboBox(hours);
timeInformation.add(endHour);
```

```
        JComboBox endMinute = new JComboBox(minutes);
        timeInformation.add(endMinute);

        JComboBox endDayOrNight = new JComboBox(dayOrNight);
        timeInformation.add(endDayOrNight);
        bottomButtons.setLayout(new BoxLayout(bottomButtons, BoxLayout.X_AXIS));

        // Sets the date and time of the combobox to the current.
        monthSelect.setSelectedIndex(Calendar.getInstance().get(Calendar.MONTH));
        daySelect.setSelectedIndex(Calendar.getInstance().get(Calendar.DAY_OF_MONTH) - 1);
        startHour.setSelectedIndex(Calendar.getInstance().get(Calendar.HOUR) - 1);
        endHour.setSelectedIndex(Calendar.getInstance().get(Calendar.HOUR) - 1);
        startMinute.setSelectedIndex(Calendar.getInstance().get(Calendar.MINUTE) / 15 - 1);
        endMinute.setSelectedIndex(Calendar.getInstance().get(Calendar.MINUTE) / 15 - 1);
        startDayOrNight.setSelectedIndex(Calendar.getInstance().get(Calendar.AM_PM));
        endDayOrNight.setSelectedIndex(Calendar.getInstance().get(Calendar.AM_PM));

        JButton cancelAppointmentButton = new JButton(CANCEL);
        JButton createAppointmentButton = new JButton(CREATE_NEW_APPOINTMENT);
        bottomButtons.add(cancelAppointmentButton);
        bottomButtons.add(createAppointmentButton);

        createAppointmentButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    if(mainPurposeField.getText().trim().equals("")||
                            descriptionArea.getText().trim().equals(""))
                        throw new ArrayIndexOutOfBoundsException();

                    String purpose = mainPurposeField.getText();
                    String description = descriptionArea.getText();
                    String appointmentType = appointmentTypeField.getText();

                    Patient patient;
                    if (uniqueID != null)
                        patient = manager.findPatient(uniqueID);
                    else
                        patient = manager.getPatients().get(patientOptions.getSelectedIndex());

                    int date;
                    String day = "" + daySelect.getSelectedItem();

                    String month;

                    if (monthSelect.getSelectedIndex() + 1 > 9)
                        month = "" + (monthSelect.getSelectedIndex() + 1);
                    else
                        month = "0" + (monthSelect.getSelectedIndex() + 1);

                    String year = "" + yearChoice.getSelectedItem();

                    date = Integer.parseInt(day + month + year);

                    int startTime = Integer.parseInt((String) startHour.getSelectedItem() +
startMinute.getSelectedItem())
                            * manager.amOrPm(startDayOrNight.getSelectedIndex());

                    int endTime = Integer.parseInt((String) endHour.getSelectedItem() +
endMinute.getSelectedItem())
                            * manager.amOrPm(endDayOrNight.getSelectedIndex());

                    Doctor availiableDoc = null;
                    if (doctorOptions.getSelectedIndex() != 0) {
                        availiableDoc = manager.getDoctors().get(doctorOptions.getSelectedIndex()
- 1);
                    } else {
                        // If the user selected to find the next available doctor, it ensures
that
```

```
                        // a doctor is found without a task during that time.
                        Doctor doctor;
                        boolean available = false;
                        for (int i = 0; i < manager.getDoctors().size() && !available; i++) {
                            doctor = manager.getDoctors().get(i);
                            if (doctor.getTasks().size() == 0) {
                                available = true;
                                availiableDoc = doctor;
                            } else {
                                available = true;
                                for (int a = 0; a < doctor.getTasks().size() && available; a++) {
                                    Task task = doctor.getTasks().get(a);
                                    if (task.getDate() == date)
                                        if (Math.abs(task.getStartTime()) == Math.abs(startTime)
||

                                                Math.abs(task.getEndTime()) == Math.abs(endTime))
                                            available = false;
                                        else if (Math.abs(task.getStartTime()) <
Math.abs(startTime) &&

                                                Math.abs(task.getEndTime()) >

Math.abs(startTime))
                                            available = false;
                                        else if (Math.abs(task.getStartTime()) <
Math.abs(endTime) &&
                                                Math.abs(task.getEndTime()) > Math.abs(endTime))
                                            available = false;
                                    }
                                    if (available) {
                                        availiableDoc = doctor;
                                        available = true;
                                    }
                                }
                            }
                        }

                        if (availiableDoc != null) {
                            manager.getReceptionist().getAppointments().add(new Appointment(purpose,
description, appointmentType,availiableDoc, patient, date, startTime, endTime));
                            availiableDoc.getTasks().add
                                    (new Task(APPOINTMENT + purpose + " with " + patient.getName() +
"(" + OWNER + patient.getOwnerFirstName() + " " + patient.getOwnerLastName() + ")", description,
availiableDoc, date, startTime, endTime));

                            availiableDoc.getTasks().get(availiableDoc.getTasks().size() -
1).setUniqueID(manager.

getReceptionist().getAppointments().get(manager.getReceptionist().getAppointments().
                                    size() - 1).getUniqueID());

                        }
                        else
                        {
                            throw new NullPointerException();
                        }


                        // If the appointment form is created from the edit patient frame.
                        if (uniqueID != null) {
                            addAppointmentFrame.dispose();
                            editPatientFrame.dispose();
                            createEditPatientDisplay(uniqueID);
                            editPatientFrame.setAlwaysOnTop(true);
                            editPatientFrame.setEnabled(true);
                        }
                        // If the appointment form is created from the second tab.
                        else {
                            addAppointmentFrame.dispose();
                            mainFrame.setEnabled(true);
                            mainFrame.setAlwaysOnTop(true);
```

```
                    }
                }catch (ArrayIndexOutOfBoundsException e1) {
                    addAppointmentFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addAppointmentFrame.setAlwaysOnTop(true);
                    }
                }
                catch (NumberFormatException e2) {
                    addAppointmentFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addAppointmentFrame.setAlwaysOnTop(true);
                    }
                } catch (NullPointerException e3) {
                    addAppointmentFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, NO_DOCTORS_AVAILABLE, ERROR,
JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addAppointmentFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

        cancelAppointmentButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addAppointmentFrame.dispose();
                addAppointmentFrame.setAlwaysOnTop(false);
                if (uniqueID == null) {
                    mainFrame.setEnabled(true);
                    mainFrame.setAlwaysOnTop(true);
                } else {
                    editPatientFrame.setEnabled(true);
                    editPatientFrame.setAlwaysOnTop(true);
                }
            }
        });

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addAppointmentFrame.dispose();
                addAppointmentFrame.setAlwaysOnTop(false);
                if (uniqueID == null) {
                    mainFrame.setEnabled(true);
                    mainFrame.setAlwaysOnTop(true);
                } else {
                    editPatientFrame.setEnabled(true);
                    editPatientFrame.setAlwaysOnTop(true);
                }
            }
        };
        addAppointmentFrame.addWindowListener(exitListener);

        addAppointmentFrame
                .setSize(new Dimension(GUI.ADD_APPOINTMENT_FRAME_WIDTH,
GUI.ADD_APPOINTMENT_FRAME_HEIGHT + 250));
        addAppointmentFrame.setLocationRelativeTo(null);
        addAppointmentFrame.setVisible(true);
    }

    /**
     * Creates the buttons used to display information about all the appointments.
     *
     * @param sortBy the integer used to specify whether to sort the appointments by
     *               doctor name, patient name or date.
```

```
     */
    private void createAppointmentsDisplayFeed(int sortBy) {
        appointmentBoxes.setLayout(new BoxLayout(appointmentBoxes, BoxLayout.PAGE_AXIS));
        secondTabContents.add(appointmentBoxes, BorderLayout.CENTER);

        appointmentBoxPanels = new ArrayList<JButton>();


        ArrayList<Appointment> sortedAppointments;

        // Decides how to sort the appointments.
        switch (sortBy) {
            case SORT_BY_DATE:
                sortedAppointments = manager.getReceptionist().sortByDate();
                break;
            case SORT_BY_PATIENTS:
                sortedAppointments = manager.getReceptionist().sortByPatient();
                break;
            case SORT_BY_DOCTORS:
                sortedAppointments = manager.getReceptionist().sortByDoctor();
                break;
            default:
                sortedAppointments = manager.getReceptionist().sortByDate();
                break;
        }

        JLabel[][] appointmentLabels = new JLabel[sortedAppointments.size()][4];

        Appointment appointment;
        for (int i = 0; i < sortedAppointments.size(); i++) {
            appointment = sortedAppointments.get(i);
            appointmentBoxPanels.add(new JButton());
            appointmentBoxes.add(appointmentBoxPanels.get(i));
            appointmentBoxPanels.get(i).setName(i + "");


            appointmentBoxPanels.get(i).setLayout(new GridLayout(2, 2));
            appointmentBoxPanels.get(i).setPreferredSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 20,
50));
            appointmentBoxPanels.get(i).setMaximumSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 50,
50));

            appointmentLabels[i][0] = new JLabel(DATE +
manager.changeDateFormat(appointment.getDate()+"",DATE_FORMAT_PLAIN,DATE_FORMAT_DASH));
            appointmentBoxPanels.get(i).add(appointmentLabels[i][0]);
            appointmentLabels[i][0].setBorder(new
CompoundBorder(appointmentLabels[i][0].getBorder(),
                    new EmptyBorder(0, 10, 0, 10)));

            appointmentLabels[i][2] = new JLabel(
                    DOCTOR + ": " + appointment.getDoctor().getFirstName() + " " +
appointment.getDoctor().getLastName());
            appointmentBoxPanels.get(i).add(appointmentLabels[i][2]);
            appointmentLabels[i][2].setBorder(new
CompoundBorder(appointmentLabels[i][2].getBorder(),
                    new EmptyBorder(0, 10, 0, 10)));

            appointmentLabels[i][1] = new JLabel(TIME +
manager.returnFormattedTime(appointment.getStartTime()
                    , appointment.getEndTime()));
            appointmentBoxPanels.get(i).add(appointmentLabels[i][1]);
            appointmentLabels[i][1].setBorder(new
CompoundBorder(appointmentLabels[i][1].getBorder(),
                    new EmptyBorder(0, 10, 0, 10)));

            appointmentLabels[i][3] = new JLabel(PATIENT + ": " +
appointment.getPatient().getName());
            appointmentBoxPanels.get(i).add(appointmentLabels[i][3]);
            appointmentLabels[i][3].setBorder(new
```

```
CompoundBorder(appointmentLabels[i][3].getBorder(),
                    new EmptyBorder(0, 10, 0, 10)));


            // Allows the user to see details about an appointment, if an appointment button is
clicked.
            appointmentBoxPanels.get(i).addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    Object source = e.getSource();
                    int selectedPanel = -1;
                    for (int a = 0; a < sortedAppointments.size() && selectedPanel == -1; a++)
                        if (((JButton)
source).getName().equals(appointmentBoxPanels.get(a).getName())) {
                            selectedPanel = a;
                        }
                    createAppointmentInfoDisplay(sortedAppointments.get(selectedPanel));
                }
            });
        }

        JScrollPane pane = new JScrollPane(appointmentBoxes);
        secondTabContents.add(pane);
        pane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
        pane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
    }

    /**
     * Creates the options displayed in the third tab. These options allow the user
     * to search for the name of a medicine and add a new medicine.
     *
     * @param text the text used to search for the medicine.
     */
    private void createTopOptionsMedicine(String text) {
        JPanel bottomOptions = new JPanel();
        bottomOptions.setLayout(new FlowLayout());

        thirdTabContents.add(bottomOptions, BorderLayout.PAGE_START);

        JLabel searchLabel = new JLabel(SEARCH);
        bottomOptions.add(searchLabel);

        JTextField searchField = new JTextField(15);
        bottomOptions.add(searchField);
        if (text != null)
            searchField.setText(text);

        enterMedicineButton = new JButton(ENTER);
        bottomOptions.add(enterMedicineButton);

        enterMedicineButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String searchedText = searchField.getText();
                ArrayList<Medicine> searchedMedicines = manager.searchMedicine(searchedText);
                medicineBoxes = new JPanel();
                medicineBoxPanels = new ArrayList<JButton>();
                thirdTabContents.removeAll();
                createTopOptionsMedicine(searchedText);
                createStockManagementsDisplayFeed(searchedMedicines);
                thirdTabContents.revalidate();
            }
        });

        JButton addMedicineButton = new JButton(ADD_MEDICINE);
        bottomOptions.add(addMedicineButton);

        addMedicineButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                mainFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(false);
```

```
                createAddStockDisplay();
                enterMedicineButton.doClick();
            }
        });

    }

    /**
     * Creates the buttons used to display each of the medicine available.
     * @param sortedMedicines
     */
    private void createStockManagementsDisplayFeed(ArrayList<Medicine> sortedMedicines) {
        medicineBoxes.setLayout(new BoxLayout(medicineBoxes, BoxLayout.PAGE_AXIS));

        Collections.sort(sortedMedicines, new Comparator<Medicine>() {
            public int compare(Medicine m1, Medicine m2) {
                if (m1.getName().equals(m2.getName()))
                    return 0;

                char[] m1Char = m1.getName().toCharArray();
                char[] m2Char = m2.getName().toCharArray();

                int length = m1Char.length;
                if (m1Char.length > m2Char.length) {
                    length = m2Char.length;
                }

                for (int i = 0; i < length; i++) {
                    if (m1Char[i] < m2Char[i])
                        return -1;
                    else if (m1Char[i] > m2Char[i])
                        return 1;
                }

                return length == m1Char.length ? -1 : 1;
            }
        });
        for(int i = 0; i < sortedMedicines.size();i++)
            if(sortedMedicines.get(i).getQuantity() == 0)
                sortedMedicines.remove(i);

        medicineBoxPanels = new ArrayList<JButton>();

        for (int i = 0; i < sortedMedicines.size(); i++) {
            medicineBoxPanels.add(new JButton());
            medicineBoxes.add(medicineBoxPanels.get(i));
            medicineBoxPanels.get(i).setName(i + "");

            medicineBoxPanels.get(i).setLayout(new BorderLayout());
            medicineBoxPanels.get(i).setPreferredSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 20,
50));
            medicineBoxPanels.get(i).setMaximumSize(new Dimension(GUI.MAIN_FRAME_WIDTH - 50,
50));

            medicineBoxPanels.get(i).add(new JLabel(NAME + sortedMedicines.get(i).getName()),
BorderLayout.WEST);
            medicineBoxPanels.get(i).add(
                    new JLabel(QUANTITY + sortedMedicines.get(i).getQuantity() + "  "),
                    BorderLayout.EAST);

            medicineBoxPanels.get(i).addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    Object source = e.getSource();
                    int selectedPanel = -1;
                    for (int a = 0; a < sortedMedicines.size() && selectedPanel == -1; a++)
                        if (((JButton)
source).getName().equals(medicineBoxPanels.get(a).getName())) {
                            selectedPanel = a;
                        }
```

```
                    createEditMedicineDisplay(sortedMedicines.get(selectedPanel));
                }
            });
        }
        thirdTabContents.setPreferredSize(new Dimension(mainFrame.getSize().width,
                (int) (mainFrame.getSize().height - mainFrame.getSize().height * 0.12)));
        thirdTabContents.add(medicineBoxes, BorderLayout.CENTER);
        pane = new JScrollPane(medicineBoxes);
        thirdTabContents.add(pane);
        pane.setViewportView(medicineBoxes);
        pane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
    }

    /**
     * Creates the frame displayed if a medicine button is clicked.
     *
     * @param medicine the medicine whose button was clicked
     */
    private void createEditMedicineDisplay(Medicine medicine)
    {
        editMedicineFrame = new JFrame();
        editMedicineFrame.setAlwaysOnTop(true);
        mainFrame.setEnabled(false);
        mainFrame.setAlwaysOnTop(false);

        createEditMedicineForm(medicine);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editMedicineFrame.dispose();
                editMedicineFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };

        editMedicineFrame.addWindowListener(exitListener);
        editMedicineFrame.setSize(new Dimension(GUI.EDIT_MEDICINE_FRAME_WIDTH,
GUI.EDIT_MEDICINE_FRAME_HEIGHT));
        editMedicineFrame.setLocationRelativeTo(null);
        editMedicineFrame.setVisible(true);
    }

    /**
     * Creates the contents of the display shown if a medicine button is clicked. It
     * displays information about the medicine and allows the user to add and remove
     * certain quantities.
     *
     * @param medicine the medicine whose button was clicked
     */
    private void createEditMedicineForm(Medicine medicine) {
        JPanel totalContent = new JPanel();
        editMedicineFrame.getContentPane().add(totalContent);
        totalContent.setLayout(new BoxLayout(totalContent, BoxLayout.Y_AXIS));

        totalContent.add(new JLabel(medicine.getName()));

        JTextArea descriptionArea = new JTextArea(12, 30);
        descriptionArea.setText(medicine.getDescription());
        descriptionArea.setEditable(false);

        JScrollPane pane = new JScrollPane(descriptionArea);
        JPanel descriptionPanel = new JPanel();

        descriptionPanel.add(pane);

        totalContent.add(descriptionPanel);
```

```
        JTextField quantityField = new JTextField(2);

        JButton addMedicineButton = new JButton(ADD);
        JButton removeMedicineButton = new JButton(REMOVE);

        addMedicineButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if(quantityField.getText().trim().equals(""))
                        throw new ArrayIndexOutOfBoundsException();

                    if(Integer.parseInt(quantityField.getText()) < 0)
                        throw new InputMismatchException();

                    medicine.setQuantity(medicine.getQuantity()
                            + Integer.parseInt(quantityField.getText()));

                    editMedicineFrame.dispose();
                    editMedicineFrame.setAlwaysOnTop(false);
                    mainFrame.setAlwaysOnTop(true);
                    mainFrame.setEnabled(true);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editMedicineFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editMedicineFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException | InputMismatchException e2) {
                    editMedicineFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editMedicineFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });

        removeMedicineButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if(quantityField.getText().trim().equals(""))
                        throw new ArrayIndexOutOfBoundsException();

                    if(Integer.parseInt(quantityField.getText()) < 0)
                        throw new InputMismatchException();
                    else if (medicine.getQuantity() - Double.parseDouble(
                            quantityField.getText()) < 0)
                        throw new ArithmeticException();

                    else
                        medicine.setQuantity(medicine.getQuantity() -
                                Integer.parseInt(quantityField.getText()));

                    editMedicineFrame.dispose();
                    editMedicineFrame.setAlwaysOnTop(false);
                    mainFrame.setAlwaysOnTop(true);
                    mainFrame.setEnabled(true);
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editMedicineFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editMedicineFrame.setAlwaysOnTop(true);
```

```
                    }
                } catch (NumberFormatException | InputMismatchException e2) {
                    editMedicineFrame.setAlwaysOnTop(false);
                    int result =  JOptionPane.showConfirmDialog(null,
INCORRECT_TEXTFIELD_MESSAGE, ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editMedicineFrame.setAlwaysOnTop(true);
                    }
                }
                catch (ArithmeticException e3) {
                    editMedicineFrame.setAlwaysOnTop(false);
                    int result =  JOptionPane.showConfirmDialog(null, NOT_ENOUGH_MEDICINE, ERROR,
JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editMedicineFrame.setAlwaysOnTop(true);
                    }
                }
            }
        });


        JPanel bottomContent = new JPanel();
        bottomContent.add(new JLabel(QUANTITY));
        bottomContent.add(quantityField);

        totalContent.add(bottomContent);

        JPanel bottomButton = new JPanel();
        bottomButton.setLayout(new BorderLayout());
        bottomButton.add(addMedicineButton, BorderLayout.EAST);
        bottomButton.add(removeMedicineButton, BorderLayout.WEST);

        totalContent.add(bottomButton);
    }

    /**
     * Creates the frame displayed when the user adds a new medicine.
     */
    private void createAddStockDisplay() {
        addStockFrame = new JFrame();
        addStockFrame.setLayout(new BorderLayout());
        addStockFrame.setAlwaysOnTop(true);
        createAddStockForm();


        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                addStockFrame.dispose();
                addStockFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };

        addStockFrame.addWindowListener(exitListener);
        addStockFrame.setSize(new Dimension(GUI.ADD_MEDICINE_FRAME_WIDTH,
GUI.ADD_MEDICINE_FRAME_HEIGHT));
        addStockFrame.setLocationRelativeTo(null);
        addStockFrame.setResizable(false);
        addStockFrame.setVisible(true);
    }

    /**
     * Creates the contents of the frame displayed when the user adds a new medicine.
     */
    private void createAddStockForm() {
        JPanel informationFields = new JPanel();
        addStockFrame.getContentPane().add(informationFields, BorderLayout.CENTER);
```

```
        JPanel medicineInformation = new JPanel();
        medicineInformation.setLayout(new FlowLayout());
        addStockFrame.getContentPane().add(medicineInformation, BorderLayout.NORTH);

        JLabel medicineInformationLabel = new JLabel(MEDICINE_INFORMATION);
        medicineInformation.add(medicineInformationLabel);

        Border border = informationFields.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        informationFields.setBorder(new CompoundBorder(border, margin));

        informationFields.setLayout(new GridLayout(0, 4));

        String[] labels = {NAME, QUANTITY};
        ArrayList<JTextField> textFields = new ArrayList<JTextField>();

        for (int i = 0; i < labels.length; i++) {
            informationFields.add(new JLabel(labels[i]));
            textFields.add(new JTextField(15));
            informationFields.add(textFields.get(i));
            textFields.get(i).setMinimumSize(new Dimension(50, 50));
        }

        JPanel bottomButtons = new JPanel();
        addStockFrame.getContentPane().add(bottomButtons, BorderLayout.SOUTH);
        bottomButtons.setLayout(new BorderLayout());

        JButton cancelButton = new JButton(CANCEL);
        bottomButtons.add(cancelButton, BorderLayout.WEST);

        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                addStockFrame.dispose();
                mainFrame.setAlwaysOnTop(true);
                mainFrame.setEnabled(true);
            }
        });


        JButton addButton = new JButton(ADD);


        bottomButtons.add(addButton, BorderLayout.EAST);

        JPanel description = new JPanel();
        description.setLayout(new BoxLayout(description, BoxLayout.X_AXIS));
        JLabel medicineUsesDescLabel = new JLabel(DESCRIPTION);

        medicineUsesDescLabel
                .setBorder(new CompoundBorder(medicineUsesDescLabel.getBorder(), new
EmptyBorder(0, 20, 0, 0)));

        description.add(medicineUsesDescLabel);
        JTextArea textArea = new JTextArea(5, 5);
        description.setBorder(new CompoundBorder(textArea.getBorder(), new EmptyBorder(0, 0, 0,
20)));
        description.add(new JScrollPane(textArea));

        bottomButtons.add(description, BorderLayout.NORTH);



        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try{
                    for(int i = 0;i < textFields.size();i++)
                        if(textFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();
```

```
                    manager.getPharmacist().getMedicines().add(
                            new Medicine(textFields.get(0).getText(),
Integer.parseInt(textFields.get(1).getText()), textArea.getText()));

                    enterMedicineButton.doClick();
                    addStockFrame.dispose();
                    addStockFrame.setAlwaysOnTop(false);
                    mainFrame.setEnabled(true);
                    mainFrame.setAlwaysOnTop(true);

                }
                catch (ArrayIndexOutOfBoundsException e1) {
                    addStockFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addStockFrame.setAlwaysOnTop(true);
                    }
                }
                catch (NumberFormatException e2) {
                    addStockFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        addStockFrame.setAlwaysOnTop(true);
                    }

                }

            }
        });


    }

    /**
     * Creates the contents displayed when the fourth tab is selected.
     */
    private void createSettingDisplay() {

        fourthTabContents.setLayout(new BorderLayout());
        JPanel editSettings = new JPanel();
        fourthTabContents.add(editSettings);

        editSettings.setLayout(new BoxLayout(editSettings, BoxLayout.Y_AXIS));
        JButton editClinicalExaminations = new JButton(EDIT_LABORATORY_TESTS);
        JPanel clinicalExaminationPanel = new JPanel();
        clinicalExaminationPanel.add(editClinicalExaminations);
        clinicalExaminationPanel
                .setBorder(new CompoundBorder(clinicalExaminationPanel.getBorder(), new
EmptyBorder(20, 20, 20, 20)));

        editSettings.add(clinicalExaminationPanel);
        editClinicalExaminations.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                mainFrame.setEnabled(false);
                mainFrame.setAlwaysOnTop(false);
                createEditPossibleTestsDisplay();
            }
        });

        JButton editRooms = new JButton(EDIT_ROOMS);
        JPanel roomPanel = new JPanel();

        roomPanel.add(editRooms);
        roomPanel.setBorder(new CompoundBorder(roomPanel.getBorder(), new EmptyBorder(20, 20, 20,
20)));
```

```
        editSettings.add(roomPanel);

        editRooms.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                mainFrame.setEnabled(false);
                mainFrame.setAlwaysOnTop(false);
                createEditRoomsDisplay();
            }
        });

        JPanel maxNumberOfExaminations = new JPanel();
        maxNumberOfExaminations.add(new JLabel(MAX_PAST_EXAMINATIONS));

        JTextField maxNumberOfExaminationsField = new JTextField(2);
        maxNumberOfExaminations.add(maxNumberOfExaminationsField);
        maxNumberOfExaminationsField.setText(maximumExaminationsDisplayed+"");

        editSettings.add(maxNumberOfExaminations);

        JButton saveButton = new JButton(SAVE);
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try{
                    int max = Integer.parseInt(maxNumberOfExaminationsField.getText());
                    maximumExaminationsDisplayed = max;
                }catch(NumberFormatException e1)
                {
                    mainFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
                            ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        mainFrame.setAlwaysOnTop(true);
                    }
                }
                maxNumberOfExaminationsField.setText(maximumExaminationsDisplayed+"");
            }
        });

        maxNumberOfExaminations.add(saveButton);

    }

    /**
     * Creates the frame shown when the user clicks edit possible tests on the fourth tab.
     */
    private void createEditPossibleTestsDisplay()
    {
        editPossibleLabTestsFrame = new JFrame();
        editPossibleLabTestsFrame.setAlwaysOnTop(true);

        createEditPossibleTestsForm();

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editPossibleLabTestsFrame.dispose();
                editPossibleLabTestsFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };
        editPossibleLabTestsFrame.addWindowListener(exitListener);

        editPossibleLabTestsFrame
                .setSize(new Dimension(GUI.POSSIBLE_LAB_TESTS_WIDTH,
GUI.POSSIBLE_LAB_TESTS_HEIGHT));
        editPossibleLabTestsFrame.setLocationRelativeTo(null);
        editPossibleLabTestsFrame.setVisible(true);
```

```
    }

    /**
     * Creates the contents of the display shown when the user clicks edit possible tests on the
fourth tab.
     */
    private void createEditPossibleTestsForm() {
        JPanel total = new JPanel();
        editPossibleLabTestsFrame.getContentPane().add(total);

        total.setLayout(new BoxLayout(total, BoxLayout.X_AXIS));

        JPanel leftPanel = new JPanel();
        JPanel rightPanel = new JPanel();
        total.add(leftPanel);
        total.add(rightPanel);
        rightPanel.setLayout(new GridLayout(0, 2));
        leftPanel.setLayout(new BoxLayout(leftPanel
                , BoxLayout.Y_AXIS));


        String[] column = {""};
        String[][] data = new String[manager.getLabTestsList().length][1];

        for (int i = 0; i < data.length; i++)
            data[i][0] = manager.getLabTestsList()[i];


        JTable examinationTable = new JTable(data, column);
        JScrollPane examinationPane = new JScrollPane(examinationTable);

        leftPanel.add(examinationPane);

        JButton exitButton = new JButton(EXIT);
        JButton removeButton = new JButton(REMOVE);

        JTextField testNameField = new JTextField(14);


        JButton addTestButton = new JButton(ADD);


        rightPanel.add(new JLabel(LABORATORY_TEST_NAME));
        rightPanel.add(testNameField);
        rightPanel.add(addTestButton);
        rightPanel.add(removeButton);
        rightPanel.add(exitButton);

        rightPanel.setBorder(new CompoundBorder(rightPanel.getBorder(), new EmptyBorder(50, 0,
50, 0)));
        leftPanel.setBorder(new CompoundBorder(leftPanel.getBorder(), new EmptyBorder(10, 10, 10,
10)));

        addTestButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    if(testNameField.getText().trim().equals(""))
                        throw new ArrayIndexOutOfBoundsException();
                    manager.getLabTests().add(testNameField.getText());
                    editPossibleLabTestsFrame.dispose();
                    createEditPossibleTestsDisplay();
                }catch (ArrayIndexOutOfBoundsException e1) {
                    editPossibleLabTestsFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
                            ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editPossibleLabTestsFrame.setAlwaysOnTop(true);
```

```
                    }
                }

            }
        });

        removeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    manager.getLabTests().remove(examinationTable.getSelectedRow());
                    editPossibleLabTestsFrame.dispose();
                    createEditPossibleTestsDisplay();
                }
                catch (ArrayIndexOutOfBoundsException e1)
                {
                    editPossibleLabTestsFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null,
SELECT_LABORATORY_TEST_FIRST, ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION)
                        editPossibleLabTestsFrame.setAlwaysOnTop(true);
                }
            }
        });

        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editPossibleLabTestsFrame.dispose();
                editPossibleLabTestsFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        });

    }

    /**
     * Creates the display shown when the user clicks edit rooms in the fourth tab.
     */
    private void createEditRoomsDisplay() {
        editRoomsFrame = new JFrame();
        editRoomsFrame.setAlwaysOnTop(true);

        JPanel mainContent = new JPanel();
        editRoomsFrame.getContentPane().add(mainContent);

        JPanel leftPanel = new JPanel();
        JPanel rightPanel = new JPanel();

        mainContent.setLayout(new BoxLayout(mainContent, BoxLayout.X_AXIS));

        leftPanel.setLayout(new BoxLayout(leftPanel, BoxLayout.Y_AXIS));
        rightPanel.setLayout(new GridLayout(0, 2));

        String[] column = {""};
        String[][] data = new String[manager.getReceptionist().getRoomsList().length-2][1];

        for (int i = 0; i < data.length; i++)
            data[i][0] = manager.getReceptionist().getRoomsList()[i+2];


        JTable roomTable = new JTable(data, column);
        JScrollPane roomPane = new JScrollPane(roomTable);

        leftPanel.setBorder(new CompoundBorder(leftPanel.getBorder(), new EmptyBorder(10, 10, 0,
10)));

        leftPanel.add(roomPane);
```

```
        JButton exitButton = new JButton(EXIT);
        JButton removeButton = new JButton(REMOVE);

        ArrayList<JTextField> textFields = new ArrayList<>();

        rightPanel.add(new JLabel(ROOM_NAME));

        JTextField roomNameField = new JTextField(14);
        rightPanel.add(roomNameField);
        textFields.add(roomNameField);

        rightPanel.add(new JLabel(COST_PER_DAY));

        JTextField roomCostField = new JTextField(14);
        rightPanel.add(roomCostField);
        textFields.add(roomCostField);

        rightPanel.add(new JLabel(SPOTS_AVAILABLE));

        JTextField roomSpotsAvailiableField = new JTextField(14);
        rightPanel.add(roomSpotsAvailiableField);
        textFields.add(roomSpotsAvailiableField);


        JButton addRoomButton = new JButton(ADD);

        rightPanel.add(addRoomButton);

        rightPanel.add(removeButton);
        rightPanel.add(exitButton);


        mainContent.add(leftPanel);
        mainContent.add(rightPanel);

        rightPanel.setBorder(new CompoundBorder(rightPanel.getBorder(), new EmptyBorder(50, 0,
50, 0)));


        addRoomButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    for(int i = 0;i<textFields.size();i++)
                        if(textFields.get(i).getText().trim().equals(""))
                            throw new ArrayIndexOutOfBoundsException();

                    manager.getReceptionist().getRooms().add(new Room(roomNameField.getText(),
                            Double.parseDouble(roomCostField.getText()),
                            Integer.parseInt(roomSpotsAvailiableField.getText())));
                    editRoomsFrame.dispose();
                    createEditRoomsDisplay();
                } catch (ArrayIndexOutOfBoundsException e1) {
                    editRoomsFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, EMPTY_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editRoomsFrame.setAlwaysOnTop(true);
                    }
                } catch (NumberFormatException e2) {
                    editRoomsFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, INCORRECT_TEXTFIELD_MESSAGE,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION) {
                        editRoomsFrame.setAlwaysOnTop(true);
                    }
                }
            }
```

```
        });

        removeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    Room room =
manager.findRoom((String)roomTable.getValueAt(roomTable.getSelectedRow(),0));
                    if(room.getPatientsInRoom().size() != 0)
                        throw new IndexOutOfBoundsException();
                    manager.getReceptionist().getRooms().remove(roomTable.getSelectedRow());
                    editRoomsFrame.dispose();
                    createEditRoomsDisplay();
                }
                catch (ArrayIndexOutOfBoundsException e1)
                {
                    editRoomsFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, SELECT_ROOM_FIRST, ERROR,
JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION)
                        editRoomsFrame.setAlwaysOnTop(true);
                }
                catch (IndexOutOfBoundsException e2)
                {
                    editRoomsFrame.setAlwaysOnTop(false);
                    int result = JOptionPane.showConfirmDialog(null, REMOVE_PATIENTS_IN_ROOM,
ERROR, JOptionPane.DEFAULT_OPTION);
                    if (result == JOptionPane.OK_OPTION)
                        editRoomsFrame.setAlwaysOnTop(true);
                }
            }
        });

        exitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                editRoomsFrame.dispose();
                editRoomsFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        });


        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                editRoomsFrame.dispose();
                editRoomsFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
            }
        };

        editRoomsFrame.addWindowListener(exitListener);
        editRoomsFrame
                .setSize(new Dimension(GUI.EDIT_ROOMS_FRAME_WIDTH, GUI.EDIT_ROOMS_FRAME_HEIGHT));
        editRoomsFrame.setLocationRelativeTo(null);
        editRoomsFrame.setVisible(true);
    }

    /**
     * Creates the frame showing detailed information about an appointment. This is opened
     * if an appointment button is pressed in the second tab.
     *
     * @param appointment the appointment whose button was pressed
     */
    private void createAppointmentInfoDisplay(Appointment appointment)
    {
        showAppointmentInfoFrame = new JFrame();
```

```
        showAppointmentInfoFrame.setAlwaysOnTop(true);
        showAppointmentInfoFrame.setEnabled(true);

        createAppointmentInfoForm(appointment);

        WindowListener exitListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                showAppointmentInfoFrame.dispose();
                showAppointmentInfoFrame.setAlwaysOnTop(false);
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        };

        showAppointmentInfoFrame.addWindowListener(exitListener);
        showAppointmentInfoFrame.setSize(new Dimension(GUI.ADD_APPOINTMENT_FRAME_WIDTH,
GUI.ADD_APPOINTMENT_FRAME_HEIGHT + 250));
        showAppointmentInfoFrame.setLocationRelativeTo(null);
        showAppointmentInfoFrame.setVisible(true);
    }
    /**
     * Creates the contents of the display showing detailed information about an appointment.
     * This is opened if an appointment button is pressed in the second tab.
     *
     * @param appointment the appointment whose button was pressed
     */
    private void createAppointmentInfoForm(Appointment appointment) {

        JPanel totalPanel = new JPanel();
        showAppointmentInfoFrame.getContentPane().add(totalPanel);

        showAppointmentInfoFrame.setLayout(new BorderLayout());

        JPanel appointmentInformation = new JPanel();
        showAppointmentInfoFrame.add(appointmentInformation, BorderLayout.NORTH);

        JLabel taskInfoLabel = new JLabel(APPOINTMENT_INFORMATION);
        JButton cancelAppointmentButton = new JButton(REMOVE_APPOINTMENT);

        appointmentInformation.add(taskInfoLabel);
        appointmentInformation.add(cancelAppointmentButton);

        cancelAppointmentButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Doctor doctor = appointment.getDoctor();
                for (int a = 0; a < doctor.getTasks().size(); a++) {
                    if (doctor.getTasks().get(a).getUniqueID().equals(appointment.getUniqueID()))
                        doctor.getTasks().remove(a);
                }
                manager.getReceptionist().getAppointments().remove(appointment);
                doctorButton.doClick();
                showAppointmentInfoFrame.dispose();
                mainFrame.setEnabled(true);
                mainFrame.setAlwaysOnTop(true);
            }
        });

        JPanel topSection = new JPanel();
        JPanel shortInformation = new JPanel();
        showAppointmentInfoFrame.add(topSection, BorderLayout.CENTER);
        topSection.setLayout(new BoxLayout(topSection, BoxLayout.PAGE_AXIS));
        topSection.add(shortInformation);

        Border border = shortInformation.getBorder();
        Border margin = new EmptyBorder(20, 20, 20, 20);
        shortInformation.setBorder(new CompoundBorder(border, margin));
```

```
        shortInformation.setLayout(new GridLayout(0, 4));
        shortInformation.add(new JLabel(APPOINTMENT_TYPE));
        shortInformation.add(new JLabel(appointment.getAppointmentType()));
        shortInformation.add(new JLabel(MAIN_PURPOSE));
        JLabel mainPurpose = new JLabel(appointment.getPurpose());
        shortInformation.add(mainPurpose);
        shortInformation.add(new JLabel(DOCTOR));

        shortInformation.add(new JLabel(appointment.getDoctor().getFirstName() + " " +
appointment.getDoctor().getLastName()));

        shortInformation.add(new JLabel(PATIENT));

        shortInformation.add(new JLabel(appointment.getPatient().getName() + "(" + OWNER +
appointment.getPatient().getOwnerFirstName() +
                " " + appointment.getPatient().getOwnerLastName() + ")"));


        JPanel description = new JPanel();
        topSection.add(description);
        description.setLayout(new BoxLayout(description, BoxLayout.X_AXIS));
        description.setBorder(new CompoundBorder(description.getBorder(), margin));
        description.add(new JLabel(APPOINTMENT_DESCRIPTION));
        JTextArea descriptionArea = new JTextArea(10, 10);
        descriptionArea.setText(appointment.getDescription());
        descriptionArea.setEditable(false);
        description.add(new JScrollPane(descriptionArea), BorderLayout.PAGE_START);

        JPanel totalBottomSection = new JPanel();
        showAppointmentInfoFrame.add(totalBottomSection, BorderLayout.SOUTH);

        totalBottomSection.setLayout(new BorderLayout());

        JPanel datePanel = new JPanel();
        JPanel timePanel = new JPanel();

        datePanel.add(new JLabel(DATE));
        datePanel.add(new
JLabel(manager.changeDateFormat(appointment.getDate()+"",DATE_FORMAT_PLAIN,DATE_FORMAT_DASH)));

        timePanel.add(new JLabel(TIME));
        timePanel.add(new JLabel(manager.returnFormattedTime(appointment.getStartTime(),
appointment.getEndTime())));

        totalBottomSection.add(datePanel, BorderLayout.EAST);
        totalBottomSection.add(timePanel, BorderLayout.WEST);
    }



}




import com.sun.xml.internal.ws.policy.privateutil.PolicyUtils;

import java.io.*;
import java.lang.reflect.Array;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Iterator;

class Manager {
```

```java
private static final String TIME_FORMAT_PLAIN = "hhmm";
private static final String TIME_FORMAT_NEW = "hh:mm";


private ArrayList<Patient> patients;
private ArrayList<Doctor> doctors;
private Receptionist receptionist;
private ArrayList<String> testsPossible;
private Pharmacist pharmacist;
private Accountant accountant;


/**
 * Returns the accountant instance.
 *
 * @return the accountant instance
 */
public Accountant getAccountant() {
   return accountant;
}

public static void main(String args[]) {

   try {
      new Manager().loadFileData();
   } catch (IOException e) {

   }
}

/**
 * Constructs a new manager.
 */
public Manager() {
   patients = new ArrayList<Patient>();
   doctors = new ArrayList<Doctor>();
   receptionist = new Receptionist();
   testsPossible = new ArrayList<String>();
   pharmacist = new Pharmacist();
   accountant = new Accountant();
}

/**
 * Loads all the saved information from text files.
 *
 * @throws IOException
 */
public void loadFileData() throws IOException {
   try {
      loadPatientData();
      loadDoctorData();
      loadAppointmentData();
      loadMedicineData();
      loadRoomData();
      loadTaskData();
      loadPossibleTestsData();
   } catch (IOException e) {
   }
}

/**
 * Loads information about all the patients from a text file and constructs
 * appropriate Patient instances.
 *
 * @throws IOException
 */
private void loadPatientData() throws IOException {
   String input;
   String name;
```

```
      char sex;
      String species;
      String breed;
      String colour;
      int dateOfBirth;
      int[] age;
      double weight;
      String ownerFirstName;
      String ownerLastName;
      String ownerEmail;
      long ownerNumber;
      String ownerAddress;
      String assignedRoom;
      String uniqueID;
      int dateOfRoomAdmittance = 0;

      BufferedReader br = new BufferedReader(
            new FileReader(new File("data/patients.txt")));

      while ((input = br.readLine()) != null) {
         name = input;
         sex = br.readLine().toCharArray()[0];
         species = br.readLine();
         breed = br.readLine();
         colour = br.readLine();
         dateOfBirth = Integer.parseInt(br.readLine());
         weight = Double.parseDouble(br.readLine());
         ownerFirstName = br.readLine();
         ownerLastName = br.readLine();
         ownerEmail = br.readLine();
         ownerNumber = Long.parseLong(br.readLine());
         ownerAddress = br.readLine();
         assignedRoom = br.readLine();
         if(!assignedRoom.equals(GUI.NONE))
            dateOfRoomAdmittance = Integer.parseInt(br.readLine());

         uniqueID = br.readLine();

         patients.add(new Patient(name, sex, species, breed, colour, dateOfBirth, weight,
uniqueID,
               ownerFirstName, ownerLastName, ownerEmail, ownerNumber, ownerAddress,
assignedRoom));
         if(dateOfRoomAdmittance!=0)
            patients.get(patients.size()-1).setDateOfRoomAdmittance(dateOfRoomAdmittance);

         input = br.readLine();

         while (!input.equals("noExaminations")) {
            String[] examinations = input.split(";");
            patients.get(patients.size()-1).getRecord().enqueue(new
Examination(Integer.parseInt(examinations[0]), examinations[1], examinations[2],
                  examinations[3], examinations[4]));
            input = br.readLine();
         }
         input = br.readLine();
         while (!input.equals("noTests")) {
            String[] tests = input.split(";");
            patients.get(patients.size() - 1).getTests()
                  .add(new Test(tests[0], tests[1], Integer.parseInt(tests[2])));
            input = br.readLine();
         }
      }
      br.close();
   }

   /**
    * Loads information about all the doctors from a text file and constructs
    * appropriate Doctor instances.
    *
```

```
     * @throws IOException If an I/O error occurs
     */
    private void loadDoctorData() throws IOException {
       String firstName;
       String lastName;
       String email;
       long number;
       String uniqueID;
       String input;
       String address;

       BufferedReader br = new BufferedReader(
             new FileReader(new File("data/doctors.txt")));
       while ((input = br.readLine()) != null) {
          firstName = input;
          lastName = br.readLine();
          email = br.readLine();
          String num = br.readLine();
          System.out.print(num);
          number = Long.parseLong(num);
          address = (br.readLine());
          uniqueID = br.readLine();
          doctors.add(new Doctor(firstName, lastName, email, number,address, uniqueID));
       }
       br.close();
    }

    /**
     * Loads information about all the appointments from a text file and constructs
     * appropriate Appointment instances.
     *
     * @throws IOException
     */
    private void loadAppointmentData() throws IOException {
       String input;

       String purpose;
       String description;
       String appointmentType;
       String doctorID;
       String patientID;
       String appointmentDescription;
       int date;
       int startTime;
       int endTime;

       BufferedReader br = new BufferedReader(
             new FileReader(new File("data/appointments.txt")));
       while ((input = br.readLine()) != null) {
          purpose = input;
          description = br.readLine();
          appointmentType = br.readLine();
          doctorID = br.readLine();
          patientID = br.readLine();
          date = Integer.parseInt(br.readLine());
          startTime = Integer.parseInt(br.readLine());
          endTime = Integer.parseInt(br.readLine());
          receptionist.getAppointments().add(new Appointment(purpose, description,
appointmentType,findDoctor(doctorID),
                findPatient(patientID), date,startTime, endTime));
       }
       br.close();
    }

    /**
     * Loads information about all the medicines from a text file and constructs
     * appropriate Medicine instances.
     *
     * @throws IOException
```

```
 */
private void loadMedicineData() throws IOException {
   String input;
   String name;
   int quantity;
   String description;

   BufferedReader br = new BufferedReader(
         new FileReader(new File("data/medicines.txt")));
   while ((input = br.readLine()) != null) {
      name = input;
      quantity = Integer.parseInt(br.readLine());
      description = br.readLine();
      pharmacist.addMedicine(name, quantity,description);
   }
   br.close();
}

/**
 * Loads information about all the rooms from a text file and constructs
 * appropriate Room instances.
 *
 * @throws IOException
 */
private void loadRoomData() throws IOException {
   String input;
   String name;
   double costPerDay;
   int availiableSpots;
   String[] mainInfo;
   String[] patientIDs;
   ArrayList<Patient> patients = new ArrayList<Patient>();

   BufferedReader br = new BufferedReader(
         new FileReader(new File("data/rooms.txt")));
   while ((input = br.readLine()) != null) {
      mainInfo = input.split(";");
      name = mainInfo[0];
      costPerDay = Double.parseDouble(mainInfo[1]);
      availiableSpots = Integer.parseInt(mainInfo[2]);
      input = br.readLine();
      if (!input.equals("none")) {
         patientIDs = input.split(";");
         for (int i = 0; i < patientIDs.length; i++) {
            patients.add(findPatient(patientIDs[i]));
         }
         receptionist.getRooms().add(new Room(name, costPerDay, availiableSpots, patients));
      } else {
         receptionist.getRooms().add(new Room(name,costPerDay,availiableSpots));
      }


   }
   br.close();
}

/**
 * Loads information about all the tasks from a text file and constructs
 * appropriate Task instances.
 *
 * @throws IOException
 */
private void loadTaskData() throws IOException {
   String purpose;
   String description;
   String doctorID;
   int date;
   int startTime;
   int endTime;
```

```
   boolean completed;
   String input;

   BufferedReader br = new BufferedReader(
         new FileReader(new File("data/tasks.txt")));
   while ((input = br.readLine()) != null) {
      purpose = input;
      description = br.readLine();
      doctorID = (br.readLine());
      date = Integer.parseInt(br.readLine());
      startTime = Integer.parseInt(br.readLine());
      endTime = Integer.parseInt(br.readLine());


      findDoctor(doctorID).getTasks().add(new Task(purpose,description,findDoctor(doctorID),
            date,startTime,endTime));
   }
   br.close();
}

/**
 * Loads information about all the available laboratory tests from a text file
 * and stores them.
 *
 * @throws IOException
 */
private void loadPossibleTestsData() throws IOException{

   String input;

   BufferedReader br = new BufferedReader(
         new FileReader(new File("data/labTestTypes.txt")));
   input = br.readLine();
   while(!input.equals(GUI.NONE))
   {
      testsPossible.add(input);
      input = br.readLine();
   }
   GUI.maximumExaminationsDisplayed = Integer.parseInt(br.readLine().trim());
   br.close();
}

/**
 * Stores all the information into text files.
 */
public void storeFileData() {
   try {
      storePatientData();
      storeDoctorData();
      storeAppointmentData();
      storeMedicineData();
      storeRoomData();
      storeTaskData();
      storePossibleTestsData();
   } catch (Exception e) {
   }
}

/**
 * Stores information about all the patients from Patient objects on a text file.
 *
 * @throws IOException
 */
private void storePatientData() throws IOException {
   Iterator<Patient> iterator = patients.iterator();
   Patient patient;

   BufferedWriter br = new BufferedWriter(
         new FileWriter(new File("data/patients.txt")));
```

```
while (iterator.hasNext()) {
   patient = iterator.next();
   if (patient != patients.get(0)) {
      br.newLine();
   }
   br.write(patient.getName());
   br.newLine();
   br.write(patient.getSex());
   br.newLine();
   br.write(patient.getSpecies());
   br.newLine();
   br.write(patient.getBreed());
   br.newLine();
   br.write(patient.getColour());
   br.newLine();
   br.write(patient.getDateOfBirth() + "");
   br.newLine();
   br.write(patient.getWeight() + "");
   br.newLine();
   br.write(patient.getOwnerFirstName());
   br.newLine();
   br.write(patient.getOwnerLastName());
   br.newLine();
   br.write(patient.getOwnerEmail());
   br.newLine();
   br.write(patient.getOwnerNumber() + "");
   br.newLine();
   br.write(patient.getOwnerAddress());
   br.newLine();
   if (patient.getAssignedRoom().equals(GUI.NONE))
      br.write(GUI.NONE);
   else {
      br.write(patient.getAssignedRoom());
      br.newLine();
      br.write(patient.getDateOfRoomAdmittance()+"");
   }

   br.newLine();
   br.write(patient.getUniqueID() + "");

   br.newLine();

   Examination examination;
   for (int i = 0; i < patient.getRecord().getPastExaminations().size(); i++) {
      examination = patient.getRecord().getPastExaminations().get(i);
      br.write("" + examination.getDate());
      br.write(";");
      br.write(examination.getSymptom());
      br.write(";");
      br.write(examination.getDiagnosis());
      br.write(";");
      br.write(examination.getTreatment());
      br.write(";");
      br.write(examination.getRemarks());
      br.newLine();
   }
   br.write("noExaminations");
   br.newLine();

   Test test;
   for (int i = 0; i < patient.getTests().size(); i++) {
      test = patient.getTests().get(i);
      br.write(test.getTestType());
      br.write(";");
      br.write(test.getStatus());
      br.write(";");
      br.write(test.getDate()+"");
      br.newLine();
```

```
            }
            br.write("noTests");
        }
        br.close();
    }

    /**
     * Stores information about all the doctors from Doctor objects on a text file.
     *
     * @throws IOException
     */
    private void storeDoctorData() throws IOException {
        Iterator<Doctor> iterator = doctors.iterator();
        Doctor doctor;

        BufferedWriter br = new BufferedWriter(
                new FileWriter(new File("data/doctors.txt")));

        while (iterator.hasNext()) {
            doctor = iterator.next();
            if (doctor != doctors.get(0)) {
                br.newLine();
            }
            br.write(doctor.getFirstName());
            br.newLine();
            br.write(doctor.getLastName());
            br.newLine();
            br.write(doctor.getEmail());
            br.newLine();
            br.write(doctor.getNumber() + "");
            br.newLine();
            br.write(doctor.getAddress());
            br.newLine();
            br.write(doctor.getUniqueID() + "");
        }

        br.close();
    }

    /**
     * Stores information about all the appointments from Appointment objects on a text file.
     *
     * @throws IOException
     */
    private void storeAppointmentData() throws IOException {

        BufferedWriter br = new BufferedWriter(
                new FileWriter(new File("data/appointments.txt")));
        ArrayList<Appointment> appointments = getReceptionist().getAppointments();
        Iterator<Appointment> iterator = appointments.iterator();
        Appointment currentAppointment;

        while (iterator.hasNext()) {
            currentAppointment = iterator.next();
            if (currentAppointment != appointments.get(0)) {
                br.newLine();
            }
            br.write(currentAppointment.getPurpose());
            br.newLine();
            br.write(currentAppointment.getDescription());
            br.newLine();
            br.write(currentAppointment.getAppointmentType());
            br.newLine();
            br.write(currentAppointment.getDoctor().getUniqueID());
            br.newLine();
            br.write(currentAppointment.getPatient().getUniqueID());
            br.newLine();
            br.write(currentAppointment.getDate() + "");
            br.newLine();
```

```
            br.write(currentAppointment.getStartTime()+"");
            br.newLine();
            br.write(currentAppointment.getEndTime()+"");
        }
        br.close();
    }

    /**
     * Stores information about all the medicines from Medicine objects on a text file.
     *
     * @throws IOException
     */
    private void storeMedicineData() throws IOException {
        Iterator<Medicine> iterator = pharmacist.getMedicines().iterator();
        Medicine medicine;

        BufferedWriter br = new BufferedWriter(
                new FileWriter(new File("data/medicines.txt")));

        while (iterator.hasNext()) {
            medicine = iterator.next();
            if (medicine != pharmacist.getMedicines().get(0)) {
                br.newLine();
            }
            br.write(medicine.getName());
            br.newLine();
            br.write(medicine.getQuantity() + "");
            br.newLine();
            br.write(medicine.getDescription());
        }
        br.close();
    }

    /**
     * Stores information about all the available rooms from an array on a text file.
     *
     * @throws IOException
     */
    private void storeRoomData() throws IOException {
        Iterator<Room> iterator = receptionist.getRooms().iterator();

        Room room;

        BufferedWriter br = new BufferedWriter(
                new FileWriter(new File("data/rooms.txt")));

        while (iterator.hasNext()) {
            room = iterator.next();
            if (room != receptionist.getRooms().get(0)) {
                br.newLine();
            }
            br.write(room.getName());
            br.write(";");
            br.write(room.getCostPerDay() + "");
            br.write(";");
            br.write(room.getAvailiableSpots() + "");
            br.newLine();
            if (room.getPatientsInRoom().size()==0) {
                br.write("none");
            } else {
                for (int i = 0; i < room.getPatientsInRoom().size(); i++) {
                    if (i != 0) {
                        br.write(";");
                    }
                    br.write(room.getPatientsInRoom().get(i).getUniqueID());
                }
            }
        }
        br.close();
```

```
}

/**
 * Stores information about all the tasks from Task objects on a text file.
 *
 * @throws IOException
 */
private void storeTaskData() throws IOException {
   Iterator<Doctor> iterator = doctors.iterator();
   Doctor doctor;
   Task task;
   boolean firstIteration = true;

   BufferedWriter br = new BufferedWriter(
         new FileWriter(new File("data/tasks.txt")));


   while (iterator.hasNext()) {
      doctor = iterator.next();
      for (int i = 0; i < doctor.getTasks().size(); i++) {
         task = doctor.getTasks().get(i);
         if(!firstIteration)
            br.newLine();
         br.write(task.getPurpose());
         br.newLine();
         br.write(task.getDescription());
         br.newLine();
         br.write(task.getDoctor().getUniqueID());
         br.newLine();
         br.write("" + task.getDate());
         br.newLine();
         br.write("" + task.getStartTime());
         br.newLine();
         br.write("" + task.getEndTime());
         firstIteration = false;
      }
   }
   br.close();
}

/**
 * Stores information about all the possible laboratory tests
 * from an array on a text file.
 *
 * @throws IOException
 */
private void storePossibleTestsData() throws IOException
{
   Iterator<String> iterator = getLabTests().iterator();
   String test;

   BufferedWriter br = new BufferedWriter(
         new FileWriter(new File("data/labTestTypes.txt")));

   while (iterator.hasNext()){
      test = iterator.next();
      if(!(test.equals(getLabTests().get(0))))
      {
         br.newLine();
      }
      br.write(test);
   }
   if(getLabTests().size() != 0)
      br.newLine();
   br.write(GUI.NONE);
   br.newLine();
   System.out.print("GOTEM" + GUI.maximumExaminationsDisplayed);
   br.write(" " + GUI.maximumExaminationsDisplayed);
   br.close();
```

```
}

/**
 * Returns the various stored patients as an arraylist.
 *
 * @return the various stored patients as an arraylist.
 */
public ArrayList<Patient> getPatients() {
   return patients;
}

/**
 * Returns the pharmacist.
 *
 * @return the pharmacist
 */
public Pharmacist getPharmacist() {
   return pharmacist;
}

/**
 * Returns the receptionist responsible for...
 *
 * @return the receptionist responsible for...
 */
public Receptionist getReceptionist() {
   return receptionist;
}

/**
 * Returns the various stored doctors as an arraylist.
 *
 * @return the various stored doctors as an arraylist
 */
public ArrayList<Doctor> getDoctors() {
   return doctors;
}

/**
 * Find the respective Doctor instance given the unique ID of the doctor.
 *
 * @param doctorID the unique ID of the doctor
 * @return the respective Doctor instance
 */
public Doctor findDoctor(String doctorID) {
   Iterator<Doctor> iterator = doctors.iterator();
   Doctor doctor;

   while (iterator.hasNext()) {
      doctor = iterator.next();
      if (doctor.getUniqueID().equals(doctorID))
         return doctor;
   }
   return null;
}

/**
 * Find the respective Patient instance given the unique ID of the patient.
 *
 * @param patientID the unique ID of the patient
 * @return the respective Patient instance
 */
public Patient findPatient(String patientID) {
   Iterator<Patient> iterator = patients.iterator();
   Patient patient;

   while (iterator.hasNext()) {
      patient = iterator.next();
      if (patient.getUniqueID().equals(patientID))
```

```
            return patient;
         }
         return null;
      }

      /**
       * Returns a list of the names of all the owners in this program.
       *
       * @return a list of the names of all the owners in this program.
       */
      public String[] getOwnerList() {
         ArrayList<String> owners = new ArrayList<String>();

         Iterator<Patient> iterator = patients.iterator();
         Patient patient;

         owners.add("   ");

         while (iterator.hasNext()) {
            patient = iterator.next();
            owners.add(patient.getOwnerFirstName() + " " + patient.getOwnerLastName());
         }
         String[] ownersFinal = new String[owners.size()];
         ownersFinal = owners.toArray(ownersFinal);
         return ownersFinal;
      }

      /**
       * Find the doctors whose name contains the given search text.
       *
       * @param name the text used to search for a doctor
       * @return the instances of Doctor that include the
       * searched text in their first or last name
       */
      public ArrayList<Doctor> searchDoctor(String name) {
         ArrayList<Doctor> foundDoctors = new ArrayList<Doctor>();
         Iterator<Doctor> iterator = doctors.iterator();
         Doctor doctor;

         while (iterator.hasNext()) {
            doctor = iterator.next();
            if (doctor.getFirstName().toLowerCase().contains(name) ||
doctor.getLastName().toLowerCase().contains(name))
               foundDoctors.add(doctor);
         }

         return foundDoctors;
      }

      /**
       * Find the patients whose name contains the given search text.
       *
       * @param name the text used to search for a patient
       * @return the instances of Patient that include the
       * searched text in their name or in their owner's name or the name of the room that they are
assigned
       */
      public ArrayList<Patient> searchPatientWithRooms(String name)
      {
         ArrayList<Patient> foundPatients = new ArrayList<Patient>();
         Iterator<Patient> iterator = patients.iterator();
         Patient patient;

         while (iterator.hasNext()) {
            patient = iterator.next();
            if (patient.getName().toLowerCase().contains(name.toLowerCase())
                  || (patient.getOwnerFirstName() + patient.getOwnerLastName()).toLowerCase()
                  .contains(name.toLowerCase()) || patient.getAssignedRoom().contains(name))
               foundPatients.add(patient);
```

```
   }

   return foundPatients;
}

/**
 * Returns the instances of Medicine that contain the searched text in their name.
 *
 * @param name the text used to search for a medicine
 * @return the instances of Medicine that include the searched text
 * in their name
 */
public ArrayList<Medicine> searchMedicine(String name) {
   ArrayList<Medicine> foundMedicines = new ArrayList<Medicine>();
   Iterator<Medicine> iterator = getPharmacist().getMedicines().iterator();
   Medicine medicine;

   while (iterator.hasNext()) {
      medicine = iterator.next();
      if (medicine.getName().toLowerCase().contains(name.toLowerCase()))
         foundMedicines.add(medicine);
   }

   return foundMedicines;
}


/**
 * Returns the list of laboratory tests that can be performed as an array.
 *
 * @return the list of laboratory tests that can be performed as an array
 */
public String[] getLabTestsList() {
   String[] finalLabTestNames = new String[testsPossible.size()];
   finalLabTestNames = testsPossible.toArray(finalLabTestNames);
   return finalLabTestNames;
}


/**
 * Returns the list of the laboratory tests that can be performed as an arraylist.
 * @return the list of the laboratory tests that can be performed as an arraylist
 */
public ArrayList<String> getLabTests()
{
   return testsPossible;
}

/**
 * Returns the Room instance with the specified room name.
 *
 * @param roomName the room name of the desired Room instance
 * @return the Room instance with the specified name
 */
public Room findRoom(String roomName)
{
   Room room = null;
   for(int i = 0; i < getReceptionist().getRooms().size();i++)
      if(roomName.equals(getReceptionist().getRooms().get(i).getName())) {
         room = getReceptionist().getRooms().get(i);
      }
   return room;
}

/**
 * A negative time value indicates PM while a positive value indicates AM. Returns
 * whether the given time is AM or PM.
 *
 * @param time the time as an integer
```

```
     * @return whether the given time is AM or PM
     */
    public String AMorPM(int time) {
       if (time > 0)
          return "AM";
       else
          return "PM";
    }

    /**
     * Returns the current date in the following format: ddmmyyyy.
     * @return the current date in the following format: ddmmyyyy
     */
    public int returnDateToday()
    {
       Calendar cal = Calendar.getInstance();
       String day = cal.get(Calendar.DAY_OF_MONTH)+ "";

       String month;


       if (cal.get(Calendar.MONTH)+1 > 9)
          month = "" + (cal.get(Calendar.MONTH)+1);
       else
          month = "0" + (cal.get(Calendar.MONTH)+1);

       String year = "" + cal.get(Calendar.YEAR);

       int date = Integer.parseInt(day + month + year);

       return date;
    }

    /**
     * Changes the format of the given date from a specified old format into a new format.
     *
     * @param date the date in the format oldFormat
     * @param oldFormat the current format of the given date
     * @param newFormat the new format which the date will be converted into
     * @return the date in the new format
     */
    public String changeDateFormat(String date,String oldFormat, String newFormat)
    {
       if(date.length() == 7)
          date = "0" + date;

       SimpleDateFormat simpleDateFormat = new SimpleDateFormat(oldFormat);
       Date newDate = null;
       try {
          newDate = simpleDateFormat.parse(date);
       } catch (ParseException e1) {
       }
       simpleDateFormat.applyPattern(newFormat);
       return (simpleDateFormat.format(newDate));
    }

    /**
     * Returns the specified start and endtime in the following format:
     * hh:mm to hh:mm.
     *
     * @param startTime the start time
     * @param endTime   the end time
     * @return the formatted start and end time
     */
    public String returnFormattedTime(int startTime, int endTime) {

       int startTimePositive = Math.abs(startTime);
       int endTimePositive = Math.abs(endTime);
```

```
      String finalStartTime = "", finalEndTime = "";
      try {
         // Converts the format of the time
         if (("" + startTimePositive).length() == 3)
            finalStartTime = new SimpleDateFormat(TIME_FORMAT_NEW).
                  format(new SimpleDateFormat(TIME_FORMAT_PLAIN).
                        parse("0" + startTimePositive));
         else
            finalStartTime = new SimpleDateFormat(TIME_FORMAT_NEW).
                  format(new SimpleDateFormat(TIME_FORMAT_PLAIN).
                        parse("" + startTimePositive));


         if (("" + startTimePositive).length() == 3)
            finalEndTime = new SimpleDateFormat(TIME_FORMAT_NEW).
                  format(new SimpleDateFormat(TIME_FORMAT_PLAIN).
                        parse("0" + endTimePositive));
         else
            finalEndTime = new SimpleDateFormat(TIME_FORMAT_NEW).
                  format(new SimpleDateFormat(TIME_FORMAT_PLAIN).
                        parse("" + endTimePositive));
      } catch (ParseException e) {
      }


      return (finalStartTime + " " + AMorPM(startTime) + " to " +
            finalEndTime + " " + AMorPM(endTime));
   }


   /**
    * Given the index of the selected item in the AM or PM combobox, returns
    * 1 if it is AM and -1 if it is PM.
    *
    * @param index the index of the selected item in the AM or PM combobox
    * @return if index is 0, return 1, if index is 1, return -1
    */
   public int amOrPm(int index) {
      if (index == 0)
         return 1;
      else
         return -1;
   }

}



class Medicine {

    private int quantity;
    private String name;
    private String description;


    /**
     * Constructs a new Medicine object.
     *
     * @param name the name of the medicine
     * @param quantity the quantity of the medicine available
     * @param description the description of the medicine
     */
    public Medicine(String name, int quantity, String description)
    {
        this.name = name;
        this.quantity = quantity;
        this.description = description;
    }
```

```java
    /**
     * Returns the description of this medicine.
     *
     * @return the description of this medicine
     */
    public String getDescription()
    {
        return description;
    }

    /**
     * Returns the name of this medicine.
     *
     * @return the name of this medicine
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the quantity available of this medicine.
     *
     * @return the quantity available of this medicine
     */
    public int getQuantity() {
        return quantity;
    }

    /**
     * Sets the quantity of available of this medicine.
     *
     * @param quantity the new quantity of this medicine
     */
    public void setQuantity(int quantity)
    {
        this.quantity = quantity;
    }


}


import java.util.ArrayList;

public class Patient {

    private String name;
    private char sex;
    private String species;
    private String breed;
    private String colour;
    private long dateOfBirth;
    private double weight;
    private String ownerFirstName;
    private String ownerLastName;
    private String ownerEmail;
    private long ownerNumber;
    private String ownerAddress;
    private String assignedRoom;
    private PatientRecord record;
    private String uniqueID;
    private ArrayList<Test> laboratoryTests;
    private int dateOfRoomAdmittance;


    /**
     * Constructs a new Patient using the specified information.
     *
```

```
     * @param name the name of the patient
     * @param sex the sex of the patient
     * @param species the species of the patient
     * @param breed the breed of the patient
     * @param colour the colour of the patient
     * @param dateOfBirth the date of birth of the patient
     * @param weight the weight of the patient
     */
    public Patient(String name, char sex, String species, String breed, String colour,
                   long dateOfBirth,double weight) {
        this.name = name;
        this.sex = sex;
        this.species = species;
        this.breed = breed;
        this.colour = colour;
        this.dateOfBirth = dateOfBirth;
        this.weight = weight;
        record = new PatientRecord();
        uniqueID = new Receptionist().generateID();
        laboratoryTests = new ArrayList<Test>();
    }

    /**
     * Constructs a new Patient using the specified information.
     *
     * @param name the name of this patient
     * @param sex the sex of this patient
     * @param species the species of this patient
     * @param breed the breed of this patient
     * @param colour the colour of this patient
     * @param dateOfBirth the date of birth of this patient
     * @param weight the weight of this patient
     * @param uniqueID the unique ID of this patient
     * @param ownerFirstName the first name of this patient's owner
     * @param ownerLastName the last name of this patient's owner
     * @param ownerEmail the email of this patient's owner
     * @param ownerNumber the telephone number of this patient's owner
     * @param ownerAddress the address of this patient's owner
     * @param assignedRoom the room assigned to this patient
     */
    public Patient(String name, char sex, String species, String breed, String colour,
                   long dateOfBirth,double weight, String uniqueID, String ownerFirstName, String
ownerLastName, String ownerEmail
            ,long ownerNumber,String ownerAddress,String assignedRoom)
    {
        this.name = name;
        this.sex = sex;
        this.species = species;
        this.breed = breed;
        this.colour = colour;
        this.dateOfBirth = dateOfBirth;
        this.weight = weight;
        record = new PatientRecord();
        this.uniqueID = uniqueID;
        this.ownerFirstName = ownerFirstName;
        this.ownerLastName = ownerLastName;
        this.ownerEmail = ownerEmail;
        this.ownerNumber = ownerNumber;
        this.ownerAddress = ownerAddress;
        this.assignedRoom = assignedRoom;
        laboratoryTests = new ArrayList<Test>();
    }


    /**
     * Adds the specified owner information regarding this patient's owner.
     *
     * @param ownerFirstName the owner's first name
     * @param ownerLastName the owner's last name
```

```
     * @param ownerEmail the owner's email
     * @param ownerNumber the owner's telephone number
     * @param ownerAddress the owner's address
     */
    public void addOwnerInformation(String ownerFirstName,String ownerLastName, String
ownerEmail, long ownerNumber,
                                    String ownerAddress) {
        this.ownerFirstName = ownerFirstName;
        this.ownerLastName = ownerLastName;
        this.ownerEmail = ownerEmail;
        this.ownerNumber = ownerNumber;
        this.ownerAddress = ownerAddress;
    }

    /**
     * Returns the date that this patient was assigned their room. 0 is returned
     * if the patient is not assigned to a room.
     *
     * @return the date that this patient was assigned their room
     */
    public int getDateOfRoomAdmittance() {

        return dateOfRoomAdmittance;
    }

    /**
     * Sets the date that this patient is assigned their room.
     *
     * @param dateOfRoomAdmittance the date that this patient is assigned their room
     */
    public void setDateOfRoomAdmittance(int dateOfRoomAdmittance) {
        this.dateOfRoomAdmittance = dateOfRoomAdmittance;
    }

    /**
     * Returns the name of this patient.
     *
     * @return the name of this patient
     */

    public String getName() {
        return name;
    }


    /**
     * Sets the name of this patient.
     *
     * @param name the new name of this patient
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Returns the sex of this patient.
     *
     * @return the sex of this patient
     */
    public char getSex() {
        return sex;
    }

    /**
     * Sets the sex of this patient.
     *
     * @param sex the new sex of this patient
     */
    public void setSex(char sex) {
```

```
    this.sex = sex;
}

/**
 * Returns the species of this patient.
 *
 * @return the species of this patient
 */
public String getSpecies() {
    return species;
}

/**
 * Sets the species of this patient.
 *
 * @param species the new species of this patient
 */

public void setSpecies(String species) {
    this.species = species;
}

/**
 * Returns the breed of this patient.
 *
 * @return the breed of this patient
 */
public String getBreed() {
    return breed;
}

/**
 * Sets the breed of this patient.
 *
 * @param breed the new breed of this patient
 */
public void setBreed(String breed) {
    this.breed = breed;
}

/**
 * Returns the colour of this patient.
 *
 * @return the colour of this patient
 */
public String getColour() {
    return colour;
}

/**
 * Sets the colour of this patient.
 *
 * @param colour the new colour of this patient
 */
public void setColour(String colour) {
    this.colour = colour;
}

/**
 * Returns the date of birth of this patient.
 *
 * @return the date of birth of this patient
 */
public long getDateOfBirth() {
    return dateOfBirth;
}

/**
 * Sets the date of birth of this patient.
```

```java
     *
     * @param dateOfBirth the new date of birth of this patient
     */
    public void setDateOfBirth(long dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    /**
     * Returns the first name of this patient's owner.
     *
     * @return the first name of this patient's owner
     */
    public String getOwnerFirstName() {
        return ownerFirstName;
    }

    /**
     * Sets the first name of this patient's owner.
     *
     * @param ownerFirstName the new first name of this patient's owner
     */
    public void setOwnerFirstName(String ownerFirstName) {
        this.ownerFirstName = ownerFirstName;
    }

    /**
     * Returns the last name of this patient's owner.
     *
     * @return the last name of this patient's owne
     */
    public String getOwnerLastName() {
        return ownerLastName;
    }

    /**
     * Sets the last name of this patient's owner.
     *
     * @param ownerLastName the new last name of this patient's owner
     */
    public void setOwnerLastName(String ownerLastName) {
        this.ownerLastName = ownerLastName;
    }

    /**
     * Returns the weight of this patient.
     *
     * @return the weight of this patient
     */
    public double getWeight() {
        return weight;
    }

    /**
     * Sets the weight of this patient.
     *
     * @param weight the new weight of this patient
     */
    public void setWeight(double weight) {
        this.weight = weight;
    }

    /**
     * Returns the email of this patient's owner.
     *
     * @return the email of this patient's owner
     */
    public String getOwnerEmail() {
        return ownerEmail;
    }
```

```java
/**
 * Sets the email of this patient's owner.
 *
 * @param ownerEmail the new email of this patient's owner
 */
public void setOwnerEmail(String ownerEmail) {
    this.ownerEmail = ownerEmail;
}

/**
 * Returns the number of this patient's owner.
 *
 * @return the number of this patient's owner
 */
public long getOwnerNumber() {
    return ownerNumber;
}

/**
 * Sets the telephone number of this patient's owner.
 *
 * @param ownerNumber the new telephone number of this patient's owner
 */
public void setOwnerNumber(long ownerNumber) {
    this.ownerNumber = ownerNumber;
}

/**
 * Returns the address of this patient's owner.
 *
 * @return the address of this patient's owner
 */
public String getOwnerAddress() {
    return ownerAddress;
}

/**
 * Sets the address of this patient's owner.
 *
 * @param ownerAddress the new address of this patient's owner
 */
public void setOwnerAddress(String ownerAddress) {
    this.ownerAddress = ownerAddress;
}

/**
 * Returns the assigned room of this patient.
 *
 * @return the assigned room of this patient
 */
public String getAssignedRoom() {
    return assignedRoom;
}

/**
 * Assigns the specified room to this patient.
 *
 * @param assignedRoom the name of the room to be assigned to this patient
 */
public void setAssignedRoom(String assignedRoom) {
    this.assignedRoom = assignedRoom;
}


/**
 * Returns this patient's record including information regarding its
 * past examinations.
 *
```

```
     * @return this patient's record including information regarding its
     * past examinations
     */
    public PatientRecord getRecord()
    {
        return record;
    }

    /**
     * Returns the unique ID of this patient.
     *
     * @return the unique ID of this patient
     */
    public String getUniqueID()
    {
        return uniqueID;
    }

    /**
     * Returns the laboratory tests performed for this patient.
     *
     * @return the laboratory tests performed for this patient
     */
    public ArrayList<Test> getTests() {
        return laboratoryTests;
    }

}


import java.util.ArrayList;

public class PatientRecord {

   private ArrayList<Examination> pastExaminations;
   private Examination head;
   private Examination rear;
   private int size;

   /**
    * Constructs a new patient record.
    */
   public PatientRecord() {
      pastExaminations = new ArrayList<Examination>();
      size = 0;
      head = null;
      rear = null;
   }

   /**
    * Returns the past examinations performed on this patient.
    *
    * @return the past examinations performed on this patient
    */
   public ArrayList<Examination> getPastExaminations() {
      Examination examination = head;
      pastExaminations = new ArrayList<Examination>();
      while (examination != null) {
         pastExaminations.add(examination);
         examination = examination.getNext();
      }


      return pastExaminations;
   }

   /**
    * Adds new examination that is performed after this examination.
    *
```

```
     * @param examination the new examination
     */
    public void enqueue(Examination examination) {
       dequeue();
       if (size == 0)
       {
          rear = head = examination;
          size++;
       }
       else {
          rear.setNext(examination);
          rear = examination;
          size++;
       }
    }

    /**
     * Removes the first examination performed in this patient record.
     */
    private void dequeue() {
       if (size == GUI.maximumExaminationsDisplayed) {
          head = head.getNext();

          size--;
       }
    }

}



import java.util.ArrayList;
import java.util.Iterator;

public class Pharmacist {

    private ArrayList medicines = new ArrayList<Medicine>();
    private Iterator<Medicine> medicineIterator = medicines.iterator();
    private Medicine currentMed;

    /**
     * Constructs a new pharmacist instance which is responsible for managing all the medicines.
     */
    public Pharmacist()
    {
        medicines = new ArrayList<Medicine>();
    }

    /**
     * Add a new medicine to the inventory.
     *
     * @param name the name of the new medicine
     * @param quantity the quantity of the new medicine
     * @param description the description of the new medicine
     */
    public void addMedicine(String name,int quantity,String description)
    {
        boolean medicineWithSameNameExists = false;
        while(medicineIterator.hasNext())
        {
            currentMed = medicineIterator.next();
            if(currentMed.getName().equals(name)) {
                currentMed.setQuantity(currentMed.getQuantity() + quantity);
                medicineWithSameNameExists = true;
            }
        }

        if(!medicineWithSameNameExists)
            medicines.add(new Medicine(name,quantity,description));
```

```
    }

    /**
     * Returns all the medicines in the inventory.
     *
     * @return all the medicines in the inventory
     */
    public ArrayList<Medicine> getMedicines() {
      return medicines;
    }

}
```

```
import java.lang.reflect.Array;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

public class Receptionist {

   private ArrayList<Appointment> appointments;
   private ArrayList<Room> rooms;


   /**
    * Constructs a new Receptionist instance.
    */
   public Receptionist() {
      appointments = new ArrayList<Appointment>();
      rooms = new ArrayList<Room>();
   }

   /**
    * Returns a list of all the rooms in the hospital.
    *
    * @return a list of all the room in the hospital
    */
   public ArrayList<Room> getRooms() {
      return rooms;
   }


   /**
    * Generates and returns a unique ID.
    *
    * @return the unique ID
    */
   public String generateID() {
      return UUID.randomUUID().toString();
   }

   /**
    * Returns all the appointments created.
    *
    * @return all the appointments created
    */
   public ArrayList<Appointment> getAppointments()
   {
      return appointments;
   }

   /**
    * Returns all the appointments of a specific patient.
    *
    * @param patient the patient whose appointments are being searched
    * @return the appointments corresponding to the specific patient
```

```
     */
    public ArrayList<Appointment> getAppointments(Patient patient) {
       ArrayList<Appointment> finalAppointments = new ArrayList<Appointment>();
       Iterator<Appointment> iterator = appointments.iterator();
       Appointment appointment;

       while(iterator.hasNext())
       {
          appointment = iterator.next();

          if(appointment.getPatient() == patient)
             finalAppointments.add(appointment);
       }
       return finalAppointments;
    }


    /**
     * Returns the list of the rooms in the hospital.
     *
     * @return the list of the rooms in the hospital.
     */
    public String[] getRoomsList() {
       String[] finalRooms = new String[rooms.size()+2];


       finalRooms[0] = GUI.NONE;
       finalRooms[1] = GUI.ASSIGN_ROOM_AUTOMATICALLY;

       for(int i = 2;i < finalRooms.length;i++)
       {
          finalRooms[i] = rooms.get(i-2).getName();
       }

       return finalRooms;
    }

    /**
     * Sorts all the appointments alphabetically based on the name of the doctor involved in the
appointment.
     *
     * @return the sorted list of appointments
     */
    public ArrayList<Appointment> sortByDoctor()
    {
       ArrayList<Appointment> sortedDoctorAppointments = appointments;
       Collections.sort(sortedDoctorAppointments, new Comparator<Appointment>() {
          public int compare(Appointment a1, Appointment a2) {
             if ((a1.getDoctor().getFirstName()+a1.getDoctor().getLastName()).
                   equals(a2.getDoctor().getFirstName()+a2.getDoctor().getLastName()))
                return 0;

             char[] a1Char =
(a1.getDoctor().getFirstName()+a1.getDoctor().getLastName()).toCharArray();
             char[] a2Char =
(a2.getDoctor().getFirstName()+a2.getDoctor().getLastName()).toCharArray();

             // Determine the shortest name.
             int length = a1Char.length;
             if (a1Char.length > a2Char.length) {
                length = a2Char.length;
             }

             for (int i = 0; i < length; i++) {
                if (a1Char[i] < a2Char[i])
                   return -1;
                else if (a1Char[i] > a2Char[i])
                   return 1;
             }
```

```
            // If all the letters of the two names are the same for the number of letters as in
the shortest name,
            // indicate that the shorter word appears first alphabetically.
            return length == a1Char.length ? -1 : 1;
         }
      });
      return sortedDoctorAppointments;
   }

   /**
    * Sorts all the appointments alphabetically based on the name of the patient involved in the
appointment.
    *
    * @return the sorted list of appointments
    */
   public ArrayList<Appointment> sortByPatient()
   {
      ArrayList<Appointment> sortedPatientAppointment = appointments;
      Collections.sort(sortedPatientAppointment, new Comparator<Appointment>() {
         public int compare(Appointment a1, Appointment a2) {
            if ((a1.getPatient().getName()).
                  equals(a2.getPatient().getName()))
               return 0;

            char[] a1Char = (a1.getPatient().getName()).toCharArray();
            char[] a2Char = (a2.getPatient().getName()).toCharArray();

            // Determine the shortest name.
            int length = a1Char.length;
            if (a1Char.length > a2Char.length) {
               length = a2Char.length;
            }

            for (int i = 0; i < length; i++) {
               if (a1Char[i] < a2Char[i])
                  return -1;
               else if (a1Char[i] > a2Char[i])
                  return 1;
            }

            // If all the letters of the two names are the same for the number of letters as in
the shortest name,
            // indicate that the shorter word appears first alphabetically.
            return length == a1Char.length ? -1 : 1;
         }
      });
      return sortedPatientAppointment;
   }


   /**
    * Sorts all the appointments based on the date and time that they take place.
    *
    * @return the sorted list of appointments
    */
   public ArrayList<Appointment> sortByDate()
   {
      ArrayList<Appointment> sortedDateAppointment = appointments;
      Collections.sort(sortedDateAppointment, new Comparator<Appointment>() {
         public int compare(Appointment a1, Appointment a2) {

            // the date of the first appointment separated into year, month and day.
            int year1 = returnDateParts(a1.getDate())[0];
            int month1 = returnDateParts(a1.getDate())[1];
            int day1 = returnDateParts(a1.getDate())[2];

            // the date of the second appointment separated into year, month and day.
            int year2 = returnDateParts(a2.getDate())[0];
```

```
            int month2 = returnDateParts(a2.getDate())[1];
            int day2  = returnDateParts(a2.getDate())[2];

            // the times of the first and second appointment.
            int finalStartTime1 = returnAMorPM(a1.getStartTime());
            int finalStartTime2 = returnAMorPM(a2.getStartTime());

            if(year1 != year2)
               return year1 - year2;
            else if(month1 != month2)
               return month1 - month2;
            else if(day1 != day2)
               return day1 - day2;
            else if(finalStartTime1 != finalStartTime2)
               return finalStartTime1 - finalStartTime2;

            return 0;
         }
      });
      return sortedDateAppointment;
   }

   /**
    * Given the date in the format ddmmyyyy, it returns it separately
    * in an array as dd, mm and yyyy.
    *
    * @param date the date in the format ddmmyyyy
    * @return the dd, mm and yyyy separated in an integer array
    */
   private int[] returnDateParts(int date)
   {
      int year1 = Integer.parseInt((date+"").substring((date+"").
            length() - 4,(date+"").length()));

      int month1 = Integer.parseInt((date+"").substring((date+"").
            length() - 6,(date+"").length()-4));

      int day1;
      if((date+"").length() == 7)
         day1 = Integer.parseInt((date+"").substring(0,1));
      else
         day1 = Integer.parseInt((date+"").substring(0,2));

      int[] fragments = {year1,month1,day1};
      return fragments;
   }


   /**
    * If the specified time is negative, then add 1200 minutes to
    * its absolute value in order to convert it to the 24-hour format.
    *
    * @param time the specified time to be converted into the 24-hour format
    * @return the specified time in the 24-hour format
    */
   private int returnAMorPM(int time)
   {
      if(time >= 0)
      {
         return Integer.parseInt(time+"");
      }
      else
         return Integer.parseInt(Math.abs(time+1200)+"");
   }
}
```

```java
import java.util.ArrayList;

public class Room {

    private int availiableSpots;
    private double costPerDay;
    private String name;
    private ArrayList<Patient> patientsInRoom;

    /**
     * Constructs a Room instance using the specified information.
     *
     * @param name            the name of this room
     * @param costPerDay      the cost per day of this room
     * @param availiableSpots the available spots in this room
     */
    public Room(String name, double costPerDay, int availiableSpots) {
        this.name = name;
        this.costPerDay = costPerDay;
        this.availiableSpots = availiableSpots;
        patientsInRoom = new ArrayList<Patient>();
    }

    /**
     * Constructs a Room instance using the specified information.
     *
     * @param name            the name of this room
     * @param costPerDay      the cost per day of this room
     * @param availiableSpots the available spots in this room
     * @param patientsInRoom  the patients currently assigned to this room
     */
    public Room(String name, double costPerDay, int availiableSpots, ArrayList<Patient>
patientsInRoom) {
        this.name = name;
        this.costPerDay = costPerDay;
        this.availiableSpots = availiableSpots;
        this.patientsInRoom = patientsInRoom;
    }

    /**
     * Returns the number of available spots in this room.
     *
     * @return the number of available spots in this room
     */
    public int getAvailiableSpots() {
        return availiableSpots;
    }

    /**
     * Returns the cost per day of this room.
     *
     * @return the cost per day of this room
     */
    public double getCostPerDay() {
        return costPerDay;
    }

    /**
     * Returns the patients in this room.
     *
     * @return the patients in this room
     */
    public ArrayList<Patient> getPatientsInRoom() {
        return patientsInRoom;
    }

    /**
     * Adds the specified patient to this room.
     *
```

```java
     * @param patient the patient to be added to this room
     */
    public void addPatientToRoom(Patient patient) {
        patientsInRoom.add(patient);
        availiableSpots--;
    }

    /**
     * Removes the specified patient from this room.
     *
     * @param patient the patient to be removed from this room
     */
    public void removePatientFromRoom(Patient patient) {
        patientsInRoom.remove(patient);
        availiableSpots++;
    }

    /**
     * Returns whether the room is full of patients or not.
     *
     * @return whether the room is full of patients or not
     */
    public boolean isFull() {
        return availiableSpots == 0;
    }


    /**
     * Returns the name of this room.
     *
     * @return the name of this room
     */
    public String getName() {
        return name;
    }


}



class Task {

   private String purpose;
   private String description;
   private Doctor doctor;
   private int date;
   private int startTime;
   private int endTime;
   private String uniqueID;

   /**
    * Constructs a new task for a doctor using the specified information.
    *
    * @param purpose the purpose of this task
    * @param description the description of this task
    * @param doctor the doctor involved in this task
    * @param date the date that this task is to be completed
    * @param startTime the time this task is meant to be started
    * @param endTime the time this task is estimated to finish
    */
   public Task(String purpose, String description, Doctor doctor, int date, int startTime,
            int endTime) {
      this.purpose = purpose;
      this.description = description;
      this.doctor = doctor;
      this.date = date;
      this.startTime = startTime;
```

```
        this.endTime = endTime;
        uniqueID = new Receptionist().generateID();
    }

    /**
     * Returns the purpose of this task.
     *
     * @return the purpose of this task
     */
    public String getPurpose() {
        return purpose;
    }

    /**
     * Returns a description of this task.
     *
     * @return the description of this task
     */
    public String getDescription() {
        return description;
    }

    /**
     * Returns the doctor involved in this task.
     *
     * @return the doctor involved in this task
     */
    public Doctor getDoctor() {
        return doctor;
    }

    /**
     * Returns the date that this task is to be completed.
     *
     * @return the date that this task is to be completed
     */
    public int getDate() {
        return date;
    }

    /**
     * Returns the time that this task is meant to be started.
     *
     * @return the time that this task is meant to be started
     */
    public int getStartTime() {
        return startTime;
    }

    /**
     * Returns the time that this task is estimated to end.
     *
     * @return the time that this task is estimated to end
     */
    public int getEndTime() {
        return endTime;
    }

    /**
     * Sets the unique ID of this task.
     *
     * @param id the new unique ID of this task
     */
    public void setUniqueID(String id)
    {
        uniqueID = id;
    }

    /**
```

```
    * Returns the unique ID of this task.
    *
    * @return the unique ID of this task
    */
   public String getUniqueID() {
      return uniqueID;
   }


}



class Test {

   private String testType;
   private String status;
   private int date;

   /**
    * Constructs a new laboratory test
    *
    * @param testType the type of test
    * @param status   the status of the test: incompleted, in progress or completed.
    * @param date     the date by which the lab test needs to be completed.
    * @param link     the link to the final lab report if has been completed already.
    */
   public Test(String testType, String status, int date) {
      super();
      this.testType = testType;
      this.status = status;
      this.date = date;
   }

   /**
    * Returns the type of this laboratory test.
    *
    * @return the type of this laboratory test
    */
   public String getTestType() {
      return testType;
   }


   /**
    * Returns the status of this laboratory test.
    *
    * @return the status of this laboratory test
    */
   public String getStatus() {
      return status;
   }

   /**
    * Changes the status of this task, if there is progress made in completing it.
    *
    * @throws NullPointerException if the status of this task states that it is already
completed, indicating
    *                              that no further progress can be made
    */
   public void changeStatus() throws NullPointerException {
      int index = 0;
      for (int i = 0; i < GUI.statusList.length; i++)
         if (status.equals(GUI.statusList[i]))
            index = i;

      if (index != GUI.statusList.length - 1) {
         status = GUI.statusList[index + 1];
```

```
      } else {
         throw new NullPointerException();
      }

   }

   /**
    * Returns the date that this laboratory test has been/ will be conducted.
    *
    * @return the date that this laboratory test has been/ will be conducted
    */
   public int getDate() {
      return date;
   }

}
```