**Drugs For You Pharmacy**

**An Automated Prescription Tracking System (PTS)**



**Authors:** Alinda Bledsoe, John Geffers, Eric Mittlefehldt, Jacob Slaght, Michelle Wesley

**Introduction**

The code demonstrated in this proposal if for the development of an automated Prescription Tracking System (PTS) for commercial use in Pharmacy Businesses. The system is designed and developed to keep track of customers, medications and their generics, and also side effects to the medication that may be prescribed to the patients.

The customer information to be recorded includes:

* First Name
* Last Name
* Telephone Number
* Date of Birth
* Insurance Provider
* Insurance Policy Number
* Prescription History

The Prescription History information to be recorded includes:

* A Unique Prescription ID Number assigned by the Pharmacy
* Medication being Prescribed
* The prescribing Physician’s name and telephone number
* Date of Issue
* Expiration Date
* Number of Refills Authorized
* Number of Units per Prescription refill (e.g. Pill, teaspoon, milliliter, etc.)
* Generic Substitute Accepted?

Medication information stocked by the pharmacy includes:

* Name of the medication
* Units of the Prescribed Medication’s dosage
* Which can serve as generic equivalents of other medications, as applicable
* Common side effects associated with the medication

The system is required to support the following queries (some will be printed out in hard-copy reports, whereas some will be view only on the screen):

* Prescription History: A full report of all prescriptions ever issued to a given customer, as requested by a given customer.
* A Report of All Side Effects associated with the prescribed medication, to be enclosed with the prescription dispensed.
* A list of generic substitutes available for a given medication
* Whether a prescription is refillable – that is, whether any refills remain and whether the prescription has yet to expire.

**Table of Contents**

**ER Diagram 4**

**Schematic Diagrams 5**

**GUI Diagrams 11**

**Main GUI 17**

**GUI Controller 21**

**GUI Stuff 27**

**Add Customer 28**

**View Customer 33**

**Search Cust GUI 38**

**New Script GUI 42**

**Script History GUI 46**

**Inventory GUI 49**

**Data File Loader 53**

**Customer Class 59**

**Customers Class 63**

**Prescription Class 66**

**Prescriptions Class 70**

**Drug Interface 73**

**Medication Class 74**

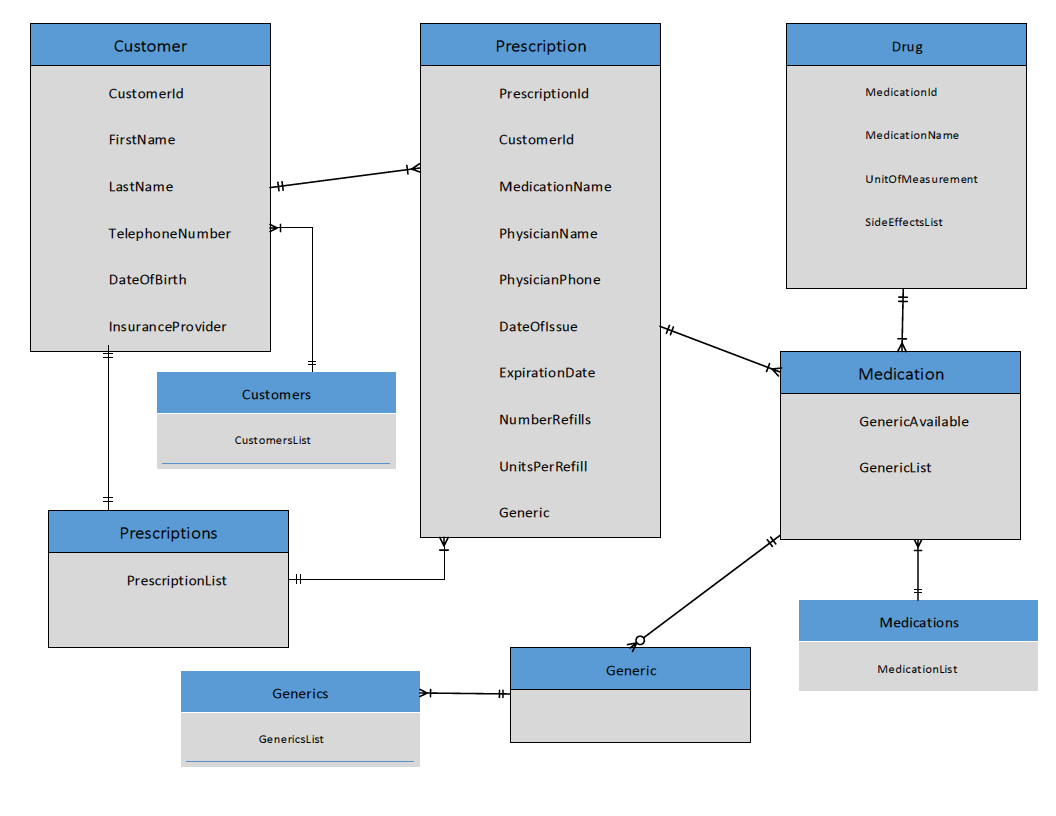
**Medications Class 77**

**Generic Class 80**

**Generics Class 82**

**Side Effects Enum 84**

**ER Diagram**

****

**Schematic Diagrams**













**GUI Diagrams**













**Main GUI**

The purpose of the Main GUI is to act as the Main menu for the program. On the Main GUI, the user will see 3 different menu items. The graphical representation of the buttons that will show up on the display (Graphical User Interface) are ”Add Customer”, “View Customer”, and “View Inventory”. These menu items are what the Pharmacist will see when he/she first accesses the software.

The “Add Customer” will route the user to the “Add Customer GUI” through the GUI Controller, where there will be fields for the pharmacist to enter into the database.

The “View Customer” will route the user to the “View Customer GUI” through the GUI Controller, where there will be a search query to locate the customer.

The “View Inventory” will route the user to the “Inventory GUI” through the GUI Controller. Once there, another menu displays options under that menu.



**Main GUI**

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

@SuppressWarnings("serial")

public class MainGUI extends JFrame {

private JPanel buttons = new JPanel();

private JButton exit = new JButton("Exit");

private JButton addCust = new JButton("Add Customer");

private JButton viewCust = new JButton("View Customers");

private JButton invMenu = new JButton("View Inventory");

private GUIController control;

The above code is the graphic representation of the buttons that will show up on the display (Graphical User Interface). This is what the Pharmacist will see when adding information into the database. As can be seen above, the user has the options of “Adding Customer”, “View Customer”, and “View Inventory”.

public MainGUI() {

createButtons();

createPanel();

add(buttons);

setSize(300, 300);

setUndecorated(true);

}

The above code is formatting the GUI appearance, including the size and the panel to add the buttons.

public MainGUI(GUIController gc) {

this();

control = gc;

}

The above code is a constructor for the GUI Controller.

private void createPanel() {

buttons.setLayout(new GridLayout(4, 1, 0, 0));

buttons.add(addCust);

buttons.add(viewCust);

buttons.add(invMenu);

buttons.add(exit);

}

The above code creates the panel, and assigns each button the class to be accessed.

private void createButtons() {

exit.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.setRun(false);

}

});

The above code adds the exit button's actionListener

addCust.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.setMainRun(false);

control.setAddRun(true);

}

});

The above code adds the add button's actionListener

viewCust.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.setMainRun(false);

control.setSearchCustRun(true);

}

});

The above code adds the view button's actionListener

invMenu.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.setMainRun(false);

control.setInvRun(true);

}

});

The above code adds the inventory button's actionListener

}

}

**GUI Controller**

The purpose of the “GUI Controller” is to act as an Air Traffic Controller. It directs the Main GUI and other GUI’s to the proper Classes. The Main GUI is the first GUI to have direct access to the GUI Controller. The menu items that are clicked on the Main GUI will be directed through there for the proper GUI associated with the button.

**GuiController**

public class GUIController {

This code creates the GUIs for the PTS

private MainGUI main;

private AddCustomerGUI addCust;

private ViewCustomerGUI viewCust;

private NewScriptGUI newScript;

private SearchCustGUI searchCust;

private ScriptHistoryGUI scriptHistory;

private InventoryGUI inventory;

The following booleans are the toggles for displaying and running of the GUI’s.

private boolean refillRun = false;

private boolean invRun = false;

private boolean run = true;

private boolean addRun = false;

private boolean viewRun = false;

private boolean mainRun = true;

private boolean newScriptRun = false;

private boolean searchCustRun = false;

private boolean scriptHistoryRun = false;

The following code creates the needed plural variables for the program: Customers, Prescriptions, and Medications.

private Customers customers;

private Prescriptions prescriptions;

private Medications medications;

public GUIController() {

main = new MainGUI(this);

addCust = new AddCustomerGUI(this);

viewCust = new ViewCustomerGUI(this);

searchCust = new SearchCustGUI(this);

scriptHistory = new ScriptHistoryGUI(this);

inventory = new InventoryGUI(this);

}

public GUIController(Customers c, Prescriptions p, Medications m) {

this();

customers = c;

prescriptions = p;

medications = m;

newScript = new NewScriptGUI(this);

updatePrescriptionHistory();

}

The updatePrescriptionHistory updates each customer's prescription.

private void updatePrescriptionHistory() {

for (int i = 0; i < customers.size(); i++) {

for (int k = 0; k < prescriptions.size(); k++) {

if (customers.getCustomerId(i).equals(

prescriptions.getCustomerId(k))) {

customers.getCustomer(i).setPrescriptionHistoryList(

prescriptions.getPrescription(k));

}

}

}

}

public boolean getRun() {

return run;

}

public void setRun(boolean r) {

run = r;

}

public Customers getCustomers() {

return customers;

}

public Prescriptions getPrescriptions() {

return prescriptions;

}

The following code adds a Customer to the Customers list.

public void addCustomers(String id, String ln, String fn, String phone,

String dob, String ins, String policy) {

customers.add(id, ln, fn, phone, dob, ins, policy);

}

The following code adds a Script to the list of Prescriptions.

public void addPrescriptions(String prescriptionId, String customerId,

String medicationName, String physicianName, String physicianPhone,

String dateOfIssue, String expirationDate, int numberRefills,

int unitsPerRefill, Boolean generic) {

prescriptions.add(prescriptionId, customerId, medicationName,

physicianName, physicianPhone, dateOfIssue, expirationDate,

numberRefills, unitsPerRefill, generic);

System.out.println(prescriptions.toString());

}

The saveCustomers() method calls the save function for the customers at one time to allow for safe usage.

public void saveCustomers() {

customers.save();

}

public boolean getNewScriptRun() {

return newScriptRun;

}

public void setNewScriptRun(boolean r) {

newScriptRun = r;

if (r == false) {

newScript.setVisible(false);

}

}

public boolean getInvRun() {

return invRun;

}

public void setInvRun(boolean r) {

invRun = r;

if (r == false) {

inventory.setVisible(false);

}

}

public boolean getRefillRun() {

return refillRun;

}

public boolean getSearchCustRun() {

return newScriptRun;

}

public void setSearchCustRun(boolean r) {

searchCustRun = r;

if (r == false) {

searchCust.setVisible(false);

}

}

public boolean getMainRun() {

return mainRun;

}

public void setMainRun(boolean r) {

mainRun = r;

if (r == false) {

main.setVisible(false);

}

}

public boolean getAddRun() {

return addRun;

}

public void setAddRun(boolean r) {

addRun = r;

if (r == false) {

addCust.setVisible(false);

}

}

public boolean getViewRun() {

return viewRun;

}

public void setViewRun(boolean r) {

viewRun = r;

if (r == false) {

viewCust.setVisible(false);

}

}

The setScriptOwner() gives the newScript GUI the proper customer of a script.

public void setScriptOwner(Customer c) {

newScript.scriptOwner(c);

}

public void setViewRun(boolean r, Customer c) {

viewRun = r;

viewCust.display(c);

if (r == false) {

viewCust.setVisible(false);

}

}

public boolean getHistoryRun() {

return newScriptRun;

}

public void setHistoryRun(boolean r) {

scriptHistoryRun = r;

if (r == false) {

scriptHistory.setVisible(false);

}

}

public void setHistoryDisplay(Customer c) {

Customer cust = c;

scriptHistory.displayHistory(cust);

}

public Medications getMedications() {

return medications;

}

The run() is called at start and calls a second method, run can be scaled.

public void run() {

checkVisibility();

}

The checkVisibility() method is just a large if-else statement that checks which GUI is active.

public void checkVisibility() {

if (searchCustRun == true) {

searchCust.setVisible(true);

}

else if (viewRun == true) {

viewCust.setVisible(true);

} else if (addRun == true) {

addCust.setVisible(true);

} else if (mainRun == true) {

main.setVisible(true);

} else if (newScriptRun == true) {

newScript.setVisible(true);

} else if (scriptHistoryRun == true) {

scriptHistory.setVisible(true);

} else if (invRun == true) {

inventory.setVisible(true);

}

}

}

**GUI Stuff**

Guistuff holds the main method for the program. It instantiates Customers, Prescriptions, Medications, Generics, and GUIController.

**GuiStuff**

public class Guistuff {

private static Customers *customers* = new Customers();

private static Prescriptions *prescriptions* = new Prescriptions();

private static Generics *generics* = new Generics();

private static Medications *medications* = new Medications(*generics*);

private static GUIController *control*;

public static void main(String[] args) {

*control* = new GUIController(*customers*, *prescriptions*, *medications*);

*control*.run();

while (*control*.getRun() != false) {

*control*.checkVisibility();

}

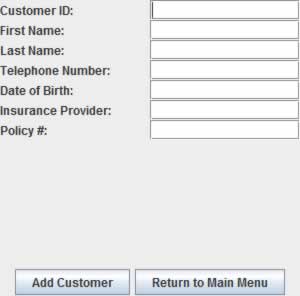
System.*exit*(0);

}

}

**Add Customer GUI**

The purpose of the Add Customer GUI is to act as the medium between the user and the database itself. The information to be added to the customer database is entered through the GUI. Once the data is entered in, provided there are no empty fields, the data is sent to the Customers Class. Fields filled out to be entered are: First Name, Last Name, Phone Number, Date of Birth, Insurance Provider, and Policy Number.



**Add Customer**

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

@SuppressWarnings("serial")

public class AddCustomerGUI extends JFrame {

private JPanel custInfo = new JPanel();

private JPanel buttons = new JPanel();

private JButton add = new JButton("Add Customer");

private JButton main = new JButton("Return to Main Menu");

private JLabel custID;

private JLabel fName;

private JLabel lName;

private JLabel telNumber;

private JLabel dateOfBirth;

private JLabel insProvider;

private JLabel polNumber;

private JTextField custIDText;

private JTextField fNameText;

private JTextField lNameText;

private JTextField telNumberText;

private JTextField dateOfBirthText;

private JTextField insProviderText;

private JTextField polNumberText;

private GUIController control;

public AddCustomerGUI() {

super("Add Customer");

createLabels();

createTextFields();

createButtons();

createPanels();

add(custInfo, BorderLayout.NORTH);

add(buttons, BorderLayout.SOUTH);

setSize(300, 300);

setUndecorated(true);

}

Initializing the GUI to Add Customers.

public AddCustomerGUI(GUIController gc) {

this();

control = gc;

}

The above code retrieves information from the GUI Controller.

private void createLabels() {

custID = new JLabel("Customer ID:");

fName = new JLabel("First Name:");

lName = new JLabel("Last Name:");

telNumber = new JLabel("Telephone Number:");

dateOfBirth = new JLabel("Date of Birth:");

insProvider = new JLabel("Insurance Provider:");

polNumber = new JLabel("Policy #:");

}

The above code is Formatting the Labels for the GUI, labeling the fields.

private void createTextFields() {

custIDText = new JTextField();

fNameText = new JTextField();

lNameText = new JTextField();

telNumberText = new JTextField();

dateOfBirthText = new JTextField();

insProviderText = new JTextField();

polNumberText = new JTextField();

}

The above code is formatting the variable text fields for input of data.

private void createButtons() {

add.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

if (custIDText.getText().equals("")

| fNameText.getText().equals("")

| lNameText.getText().equals("")

| telNumberText.getText().equals("")

| dateOfBirthText.getText().equals("")

| insProviderText.getText().equals("")

| polNumberText.getText().equals("")) {

JOptionPane.showMessageDialog(null,

"Invalid Customer Add, Please Try Again");

clearForm();

} else {

control.addCustomers(custIDText.getText(),

lNameText.getText(), fNameText.getText(),

telNumberText.getText(), dateOfBirthText.getText(),

insProviderText.getText(), polNumberText.getText());

control.saveCustomers();

control.setAddRun(false);

control.setMainRun(true);

clearForm();

}

}

});

The IF STATEMENT is to check for missing data, all fields need to be filled out. The above code is the failsafe check, if data is missing then an error will be returned, data will NOT be saved, and the user must start over.

main.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.saveCustomers();

control.setAddRun(false);

control.setMainRun(true);

clearForm();

}

});

The above code adds the exit button's actionListener.

}

private void createPanels() {

custInfo.setLayout(new GridLayout(7, 1, 0, 0));

custInfo.add(custID);

custInfo.add(custIDText);

custInfo.add(fName);

custInfo.add(fNameText);

custInfo.add(lName);

custInfo.add(lNameText);

custInfo.add(telNumber);

custInfo.add(telNumberText);

custInfo.add(dateOfBirth);

custInfo.add(dateOfBirthText);

custInfo.add(insProvider);

custInfo.add(insProviderText);

custInfo.add(polNumber);

custInfo.add(polNumberText);

buttons.add(add);

buttons.add(main);

}

The above code adds the variables to both the Customer and Customers Classes. Customer is the current customer information being entered into the system. Customers is the customer’s history, which is initialized on first entry.

private void clearForm() {

custIDText.setText("");

fNameText.setText("");

lNameText.setText("");

telNumberText.setText("");

dateOfBirthText.setText("");

insProviderText.setText("");

polNumberText.setText("");

}

}

The above code resets all variables of the customer table back to their default values.

**View Customer GUI**

The purpose of the “View Customer GUI” is to call up the customer’s information. The GUI will access the Customers Class to retrieve the information to be returned. The Customers Class will access the Customer Class to retrieve the format in which the data is to be returned. The customer’s information will be returned to the user in the proper format that will be readable for the user. Once the customer’s information is accessed and displayed, the user has the option of “New Prescription”, “Prescription History”, “Refill Prescription”, or “Return to Main Menu”.

New Prescription – Fill out a new prescription for the customer.

Prescription History – All prescriptions filled by the pharmacist for the said customer.

Refill Prescription – Refills the existing prescription for the customer.



**View Customer GUI**

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

@SuppressWarnings("serial")

public class ViewCustomerGUI extends JFrame

{

private JLabel name = new JLabel();

private JLabel phone =new JLabel();

private JLabel dateOfBirth =new JLabel();

private JLabel provider =new JLabel();

private JLabel policy =new JLabel();

private JButton newScript = new JButton("New Prescription");

private JButton drugHistory = new JButton("Prescription History");

private JButton fillScript = new JButton("Refill Prescription");

private JButton main = new JButton("Return to Main Menu");

private JPanel info = new JPanel();

private JPanel buttons = new JPanel();

private JPanel subInfo = new JPanel();

private Customer cust;

private GUIController control;

public ViewCustomerGUI()

{

createLabels();

createButtons();

createPanels();

add(info,BorderLayout.NORTH);

add(subInfo,BorderLayout.CENTER);

add(buttons,BorderLayout.SOUTH);

setSize(300,300);

setUndecorated(true);

}

The above code is initializing the format of the GUI.

public ViewCustomerGUI(GUIController gc)

{

this();

control = gc;

}

The above code retrieves information from the GUI Contorller.

private void createLabels()

{

name.setText("Customer Last Name,Customer First Name");

phone.setText("Phone: 555-555-5555");

dateOfBirth.setText("DOB: 01/01/1970");

provider.setText("Provider: Provider");

policy.setText("Policy Number: 12345");

}

The above code is setting the text fields to their default values.

private void createButtons()

{

newScript.addActionListener(

new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

control.setViewRun(false);

control.setScriptOwner(cust);

control.setNewScriptRun(true);

}

}); //Adds the exit button's actionListener

main.addActionListener(

new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

control.setViewRun(false);

control.setMainRun(true);

}

});

The above code adds the exit button's actionListener.

drugHistory.addActionListener(

new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

control.setHistoryDisplay(cust);

control.setHistoryRun(true);

control.setViewRun(false);

}

});

The above code adds the exit button's actionListener.

fillScript.addActionListener(

new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

//Not added

}

});

The above code adds the exit button's actionListener.

}

private void createPanels() {

subInfo.setLayout(new GridLayout(2,2,0,0));

info.add(name);

subInfo.add(phone);

subInfo.add(dateOfBirth);

subInfo.add(provider);

subInfo.add(policy);

buttons.setLayout(new GridLayout(2,2,0,0));

buttons.add(newScript);

buttons.add(drugHistory);

buttons.add(fillScript);

buttons.add(main);

}

The above code is formatting the panel and adding the values that are to be returned.

public void display(Customer c) {

cust = c;

name.setText(c.getLastName()+ "," + cust.getFirstName());

phone.setText("Phone: " + cust.getTelephoneNumber());

dateOfBirth.setText("DOB: "+ cust.getDateOfBirth());

provider.setText("Provider: " + cust.getInsuranceProvider());

policy.setText("Policy Number: " + cust.getPolicyNumber());

}

}

The above code is formatting the data to be returned.

**Search Cust GUI**

The purpose of the “Search Customer GUI” is to search the customer’s information in the database, based on the first and last name. The GUI will access the Customer database and look for a hit matching the first and last name entered into the search. The search will result in a hit based on the search criteria. Provided there is a hit, this will automatically launch the “View Customer GUI”.



**Search Cust GUI**

import java.awt.BorderLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

@SuppressWarnings("serial")

public class SearchCustGUI extends JFrame

{

private JLabel searchFirst = new JLabel();

private JLabel searchLast = new JLabel();

private JPanel searchCust = new JPanel();

private JTextField fNameText;

private JTextField lNameText;

private JPanel buttons = new JPanel();

private JButton submit = new JButton("Submit Search Query");

private JButton main = new JButton("Return to Main Menu");

private GUIController control;

The above code is formatting the GUI to do a query, looking for any hits that would fit the criteria filled out on the GUI.

public SearchCustGUI()

{

createLabels();

createTextFields();

createButtons();

createPanels();

add(searchCust,BorderLayout.CENTER);

add(buttons,BorderLayout.SOUTH);

setSize(300,300);

setUndecorated(true);

}

public SearchCustGUI(GUIController c)

{

this();

control = c;

}

The above code retrieves information from the GUI Controller.

private void createLabels()

{

searchFirst.setText("Customer First Name");

searchLast.setText("Customer Last Name");

}

private void createTextFields()

{

fNameText= new JTextField();

lNameText= new JTextField();

}

private void createButtons()

{

submit.addActionListener(

new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

if((control.getCustomers().contains(lNameText.getText(),fNameText.getText())==true &&(control.getCustomers().getCustomer(lNameText.getText(),fNameText.getText())!=null)))

The IF STATEMENT is to check for missing data, all fields need to be filled out and not NUll. The above code is the failsafe check, if data is missing then an error will be returned, data will NOT be saved, and the user must start over.

{

control.setSearchCustRun(false);

Customer c = control.getCustomers().getCustomer(lNameText.getText(),fNameText.getText());

control.setViewRun(true,c);

lNameText.setText("");

fNameText.setText("");

}

else

{

JOptionPane.showMessageDialog(null, "Customer Not Found, Please Try Again");

lNameText.setText("");

fNameText.setText("");

}

}

});

The above code adds the exit button's actionListener

main.addActionListener(

new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

lNameText.setText("");

fNameText.setText("");

control.setSearchCustRun(false);

control.setMainRun(true);

}

});

The above code adds the exit button's actionListener.

}

private void createPanels()

{

searchCust.setLayout(new GridLayout(2,2));

// searchCust.setLayout(new GridLayout(2,2,0,0));

searchCust.add(searchLast);

searchCust.add(lNameText);

searchCust.add(searchFirst);

searchCust.add(fNameText);

buttons.setLayout(new GridLayout(1,2,0,0));

buttons.add(submit);

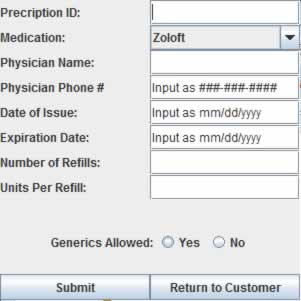
buttons.add(main);

}

}

**New Script GUI**

The purpose of the “New Script GUI” is to add a new prescription into the database for the customer. The GUI will prompt the pharmacist to enter the Medication to be prescribed, the name of the Physician who prescribed the medication, the telephone number of the physician, the date the prescription was issued by the physician, the medication’s expiration date, the number of refills, and the amount to be refilled. This information is entered into the system and stored in the database. First prescriptions for customers will automatically start a prescription history for the customer.



**New Script GUI**

import java.awt.BorderLayout;

import java.awt.FlowLayout;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

@SuppressWarnings("serial")

public class NewScriptGUI extends JFrame {

private JPanel container = new JPanel();

private JPanel generic = new JPanel();

private JPanel buttons = new JPanel();

private JPanel script = new JPanel();

@SuppressWarnings("rawtypes")

private JComboBox medsList;

private Customer cust;

private JRadioButton rYes = new JRadioButton("Yes");

private JRadioButton rNo = new JRadioButton("No");

private ButtonGroup radios = new ButtonGroup();

private JLabel l1 = new JLabel("Precription ID:");

private JLabel l2 = new JLabel("Medication:");

private JLabel l3 = new JLabel("Physician Name:");

private JLabel l4 = new JLabel("Physician Phone #");

private JLabel l5 = new JLabel("Date of Issue:");

private JLabel l6 = new JLabel("Expiration Date:");

private JLabel l7 = new JLabel("Number of Refills:");

private JLabel l8 = new JLabel("Units Per Refill:");

private JLabel l9 = new JLabel("Generics Allowed:");

private JTextField scriptIdText;

private JTextField medicationText;

private JTextField physicianText;

private JTextField physicianPhoneText;

private JTextField dateOfIssueText;

private JTextField expirationDateText;

private JTextField numRefillsText;

private JTextField unitRefillsText;

private JButton submit = new JButton("Submit");

private JButton customer = new JButton("Return to Customer");

private boolean isGeneric;

private GUIController control;

public NewScriptGUI() {

createButtons();

add(container, BorderLayout.NORTH);

add(buttons, BorderLayout.SOUTH);

setSize(300, 300);

setUndecorated(true);

}

public NewScriptGUI(GUIController c) {

this();

control = c;

createComboBox();

createTextFields();

createPanel();

}

private void createTextFields() {

scriptIdText = new JTextField();

medicationText = new JTextField();

physicianText = new JTextField();

physicianPhoneText = new JTextField();

physicianPhoneText.setText("Input as ###-###-####");

dateOfIssueText = new JTextField();

dateOfIssueText.setText("Input as mm/dd/yyyy");

expirationDateText = new JTextField();

expirationDateText.setText("Input as mm/dd/yyyy");

numRefillsText = new JTextField();

unitRefillsText = new JTextField();

}

@SuppressWarnings({ "rawtypes", "unchecked" })

private void createComboBox() {

Medications meds = control.getMedications();

String[] medName = new String[meds.size() - 1];

for (int i = 0; i < meds.size() - 1; i++) {

medName[i] = meds.getMedications().get(i).getMedicationName();

}

medsList = new JComboBox(medName);

}

private void createPanel() {

container.setLayout(new GridLayout(2, 1, 0, 0));

script.setLayout(new GridLayout(9, 2, 0, 0));

generic.setLayout(new FlowLayout());

buttons.setLayout(new GridLayout(1, 2, 0, 0));

generic.add(l9);

generic.add(rYes);

generic.add(rNo);

script.add(l1);

script.add(scriptIdText);

script.add(l2);

script.add(medsList);

script.add(l3);

script.add(physicianText);

script.add(l4);

script.add(physicianPhoneText);

script.add(l5);

script.add(dateOfIssueText);

script.add(l6);

script.add(expirationDateText);

script.add(l7);

script.add(numRefillsText);

script.add(l8);

script.add(unitRefillsText);

container.add(script);

container.add(generic);

buttons.add(submit);

buttons.add(customer);

}

public void scriptOwner(Customer c) {

cust = c;

}

private void createButtons() {

The following code is for Radio Buttons.

rYes.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

isGeneric = true;

}

});

rNo.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

isGeneric = false;

}

});

radios.add(rYes);

radios.add(rNo);

The above code is for Solid Buttons.

submit.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

cust.getPrescriptionHistoryList().add(

new Prescription(scriptIdText.getText(),

cust.getCustomerId(),

medicationText.getText(),

physicianText.getText(),

physicianPhoneText.getText(), dateOfIssueText.getText(),

expirationDateText.getText(),

Integer.parseInt(numRefillsText.getText()),

Integer.parseInt(unitRefillsText.getText()),

isGeneric));

control.setNewScriptRun(false);

control.setViewRun(true, cust);

The above code adds the Script to the Customer's History.

}

});

The above code adds the exit button's actionListener.

customer.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.setNewScriptRun(false);

control.setViewRun(true, cust);

}

});

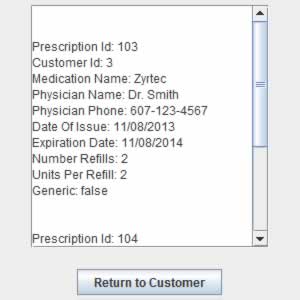
The above code adds the exit button's actionListener.

}

}

**Script History GUI**

The purpose of the “Script History GUI” is to view the prescription history for a specific customer in the database. The GUI will search the database to verify the record exists. The customer’s prescription history will then be returned back to the pharmacist in the proper format.



**Script History GUI**

import javax.swing.\*;

import java.awt.BorderLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

@SuppressWarnings("serial")

public class ScriptHistoryGUI extends JFrame {

private GUIController control;

private JPanel buttons = new JPanel();

private JPanel history = new JPanel();

private JButton view = new JButton("Return to Customer");

private JTextArea displayHistory;

private JScrollPane pane;

public ScriptHistoryGUI() {

createTextArea();

createPanels();

createButtons();

add(history, BorderLayout.NORTH);

add(buttons, BorderLayout.SOUTH);

setSize(300, 300);

setUndecorated(true);

}

public ScriptHistoryGUI(GUIController c) {

this();

control = c;

}

The above code is retrieving the format from the GUI Controller.

private void createTextArea() {

displayHistory = new JTextArea(15, 20);

displayHistory.setEditable(false);

pane = new JScrollPane(displayHistory);

}

private void createPanels() {

history.add(pane);

// history.add(displayHistory);

history.setVisible(true);

buttons.add(view);

}

private void createButtons() {

view.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae) {

control.setHistoryRun(false);

control.setViewRun(true);

}

});

The above code adds the exit button's actionListener

}

public void displayHistory(Customer c) {

displayHistory.setText("");

Customer cust = c;

if (cust.getPrescriptionHistoryList().size() < 1) {

displayHistory.append(cust.getLastName() + ","

+ cust.getFirstName() + " has no Prescription History\n\n");

} else {

for (int i = 0; i < cust.getPrescriptionHistoryList().size(); i++) {

displayHistory.append(cust.getPrescriptionHistoryList().get(i)

.toString());

}

}

}

}

**Inventory GUI**

The purpose of the “Inventory GUI” is to call up the medication that is carried by the pharmacy. Medication can be added to the database or viewed in this GUI. The “Add Medication” feature will prompt the pharmacist to enter the Medication name, the Unit of Measurement, and possible Side Effects. The user will also be asked if there is a generic available for the medication. This information will be entered into the database. The “View Inventory “will allow the user to see what medications are carried by the pharmacy.

**Inventory GUI**

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\*;

@SuppressWarnings("serial")

public class InventoryGUI extends JFrame

{

private JPanel buttons = new JPanel();

private JButton main = new JButton("Return to Main");

private JButton addMed = new JButton("Add Medication");

private JButton viewInv = new JButton("View Inventory");

private GUIController control;

The above code is formatting the GUI to “Return to Main Menu”, “Add Medication”, and to “View Inventory”.

public InventoryGUI() {

createButtons();

createPanel();

add(buttons);

setSize(300,300);

setUndecorated(true);

}

public InventoryGUI(GUIController c) {

this();

control = c;

}

The above code retrieves information from the GUI Controller.

private void createPanel() {

buttons.setLayout(new GridLayout(3,1,0,0));

buttons.add(addMed);

buttons.add(viewInv);

buttons.add(main);

}

The above code is formatting the GUI to retrieve information.

private void createButtons() {

main.addActionListener( new ActionListener()

{

@Override

public void actionPerformed(ActionEvent ae)

{

control.setInvRun(false);

control.setMainRun(true);

}

});

The above code adds the exit button's actionListener.

addMed.addActionListener( new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae)

{

JOptionPane.showMessageDialog(null, "Not currently functioning, select a differnt option.");

}

});

The above code adds the add button's actionListener.

viewInv.addActionListener( new ActionListener() {

@Override

public void actionPerformed(ActionEvent ae)

{

JOptionPane.showMessageDialog(null, "Not currently functioning, select a differnt option.");

}

});

The above code adds the view button's actionListener.

}

}

**DataFileLoader**

The data file loader finds the data in file or database and brings it to the desired location. Such as finding the customer, if there is a generic available, or even a customer’s prescription history. It loads customers from a text file and puts it into the customers class. It saves to the text file information that is added. It also saves and loads to other files if needed, such as prescription history or medications.

**Data File Loader**

import java.io.\*;

import java.util.\*;

public class DataFileLoader {

public ArrayList<Customer> getCustomers() throws Exception {

BufferedReader customerFile = new BufferedReader(new FileReader(file

+ "Customer.txt"));

ArrayList<Customer> customers = new ArrayList<Customer>();

Customer aCustomer;

String dataRow = customerFile.readLine(); // Read first line.

The while checks to see if the data is null. If it is, we've hit the end of the file. If not, process the data.

while (dataRow != null) {

String[] dataArray = dataRow.split("\t");

aCustomer = new Customer();

aCustomer.setCustomerId(dataArray[0]);

aCustomer.setFirstName(dataArray[1]);

aCustomer.setLastName(dataArray[2]);

aCustomer.setTelephoneNumber(dataArray[3]);

aCustomer.setDateOfBirth(dataArray[4]);

aCustomer.setInsuranceProvider(dataArray[5]);

aCustomer.setPolicyNumber(dataArray[6]);

customers.add(aCustomer);

dataRow = customerFile.readLine(); // Read next line of data.

}

The above code reads the next line of data.

The following code closes the file once all data has been read.

customerFile.close();

return customers;

}

public void saveCustomers(ArrayList<Customer> aCustomers) throws Exception {

BufferedWriter customerFile = new BufferedWriter(new FileWriter(file

+ "Customer.txt"));

ArrayList<Customer> customers = aCustomers;

for (Customer customer : customers) {

String customerStr = "";

customerStr = customer.getCustomerId() + "\t"

+ customer.getFirstName() + "\t" + customer.getLastName()

+ "\t" + customer.getTelephoneNumber() + "\t"

+ customer.getDateOfBirth() + "\t"

+ customer.getInsuranceProvider() + "\t"

+ customer.getPolicyNumber();

customerFile.write(customerStr);

customerFile.newLine();

}

The following code closes the file once all data has been read.

customerFile.close();

}

public ArrayList<Prescription> getPrescriptions() throws Exception {

BufferedReader prescriptionFile = new BufferedReader(new FileReader(

file + "Presciption.txt"));

ArrayList<Prescription> prescriptions = new ArrayList<Prescription>();

Prescription aPrescription;

String dataRow = prescriptionFile.readLine();

The above code reads the first line.

The while checks to see if the data is null. If it is, we've hit the end of the file. If not, process the data.

while (dataRow != null) {

String[] dataArray = dataRow.split("\t");

aPrescription = new Prescription();

aPrescription.setPrescriptionId(dataArray[0]);

aPrescription.setCustomerId(dataArray[1]);

aPrescription.setMedicationName(dataArray[2]);

aPrescription.setPhysicianName(dataArray[3]);

aPrescription.setPhysicianPhone(dataArray[4]);

aPrescription.setDateOfIssue(dataArray[5]);

aPrescription.setExpirationDate(dataArray[6]);

aPrescription.setNumberRefills(Integer.parseInt(dataArray[7]));

aPrescription.setUnitsPerRefill(Integer.parseInt(dataArray[8]));

aPrescription.setGeneric(Boolean.parseBoolean(dataArray[9]));

prescriptions.add(aPrescription);

dataRow = prescriptionFile.readLine(); // Read next line of data.

}

The following code closes the file once all data has been read.

prescriptionFile.close();

return prescriptions;

}

public ArrayList<Generic> getGenerics() throws Exception {

BufferedReader genericFile = new BufferedReader(new FileReader(file

+ "Generic.txt"));

ArrayList<Generic> generics = new ArrayList<Generic>();

Generic aGeneric;

String dataRow = genericFile.readLine();

while (dataRow != null) {

String[] dataArray = dataRow.split("\t");

aGeneric = new Generic();

aGeneric.setMedicationId(dataArray[0]);

aGeneric.setMedicationName(dataArray[1]);

aGeneric.setUnitOfMeasurement(dataArray[2]);

for (int i = 3; i < dataArray.length; i++) {

aGeneric.addSideEffects(SideEffects.valueOf(dataArray[i]));

}

generics.add(aGeneric);

dataRow = genericFile.readLine();

}

genericFile.close();

return generics;

}

public ArrayList<Medication> getMedications(Generics generics)

throws Exception {

BufferedReader medicationFile = new BufferedReader(new FileReader(file

+ "Medication.txt"));

ArrayList<Medication> medications = new ArrayList<Medication>();

Medication aMedication;

String dataRow = medicationFile.readLine(); // Read first line.

The while checks to see if the data is null. If it is, we've hit the end of the file. If not, process the data.

while (dataRow != null) {

String[] dataArray = dataRow.split("\t");

aMedication = new Medication();

aMedication.setMedicationId(dataArray[0]);

aMedication.setMedicationName(dataArray[1]);

aMedication.setUnitOfMeasurement(dataArray[2]);

aMedication

.setGenericsAvailable(Boolean.parseBoolean(dataArray[3]));

if (aMedication.getGenericsAvailable()) {

String[] genericsString = dataArray[4].split("|");

for (int x = 0; x < genericsString.length;) {

for (int k = 0; k < generics.size(); k++) {

if (generics.getId(k).equals(genericsString[x])) {

aMedication.addGeneric(generics.getGeneric(k));

}

}

x++;

}

for (int i = 5; i < dataArray.length; i++) {

aMedication.addSideEffects(SideEffects

.valueOf(dataArray[i]));

}

} else {

for (int i = 4; i < dataArray.length; i++) {

aMedication.addSideEffects(SideEffects

.valueOf(dataArray[i]));

}

}

medications.add(aMedication);

dataRow = medicationFile.readLine();

The above code reads the next line of data.

}

The following code closes the file once all data has been read.

medicationFile.close();

return medications;

}

The following code writes and saves the data.

public void savePrescriptions(ArrayList<Prescription> aPrescriptions)

throws Exception {

BufferedWriter prescriptionFile = new BufferedWriter(new FileWriter(

file + "Presciption.txt"));

ArrayList<Prescription> prescriptions = aPrescriptions;

for (Prescription prescription : prescriptions) {

String prescriptionStr = "";

prescriptionStr = prescription.getPrescriptionId() + "\t"

+ prescription.getCustomerId() + "\t"

+ prescription.getMedicationName() + "\t"

+ prescription.getPhysicianName() + "\t"

+ prescription.getPhysicianPhone() + "\t"

+ prescription.getDateOfIssue() + "\t"

+ prescription.getExpirationDate() + "\t"

+ prescription.getNumberRefills() + "\t"

+ prescription.getUnitsPerRefill() + "\t"

+ prescription.getGeneric();

prescriptionFile.write(prescriptionStr);

prescriptionFile.newLine();

}

The following code closes the file once all data has been read.

prescriptionFile.close();

}

}

**Customer**

The customer class is a blueprint for every customer that is put into the system. It holds their customer Id, first and last name, telephone number, date of birth, insurance provider, policy number and their prescription history.

**Customer**

import java.util.ArrayList;

public class Customer {

private String customerId;

private String lastName;

private String firstName;

private String telephoneNumber;

private String dateOfBirth;

private String insuranceProvider;

private String policyNumber;

private ArrayList<Prescription> prescriptionHistoryList = new ArrayList<Prescription>();

The above code is defining the variables and setting up the Prescription History as an Array.

public void setCustomerId(String aCustomerId) {

this.customerId = aCustomerId;

}

public void setLastName(String aLastName) {

this.lastName = aLastName;

}

public void setFirstName(String aFirstName) {

this.firstName = aFirstName;

}

public void setTelephoneNumber(String aTelephoneNumber) {

this.telephoneNumber = aTelephoneNumber;

}

public void setDateOfBirth(String aDateOfBirth) {

this.dateOfBirth = aDateOfBirth;

}

public void setInsuranceProvider(String aInsuranceProvider) {

this.insuranceProvider = aInsuranceProvider;

}

public void setPolicyNumber(String aPolicyNumber) {

this.policyNumber = aPolicyNumber;

}

public void setPrescriptionHistoryList(Prescription aPrescription) {

this.prescriptionHistoryList.add(aPrescription);

}

The above statements are for field population. Each of the fields is being inputted into the database.

public String getCustomerId() {

return customerId;

}

public String getLastName() {

return lastName;

}

public String getFirstName() {

return firstName;

}

public String getTelephoneNumber() {

return telephoneNumber;

}

public String getDateOfBirth() {

return dateOfBirth;

}

public String getInsuranceProvider() {

return insuranceProvider;

}

public String getPolicyNumber() {

return policyNumber;

}

public ArrayList<Prescription> getPrescriptionHistoryList() {

return this.prescriptionHistoryList;

}

The following is retrieving from the GUI, the inputted values for the table.

public Customer() {

This is the empty constructor.

}

public Customer(String customerId, String lastName, String firstName,

String telephoneNumber, String dateOfBirth,

String insuranceProvider, String policyNumber) {

this.customerId = customerId;

this.lastName = lastName;

this.firstName = firstName;

this.telephoneNumber = telephoneNumber;

this.dateOfBirth = dateOfBirth;

this.insuranceProvider = insuranceProvider;

this.policyNumber = policyNumber;

}

public void setCustomerInfo(String aCustomerId, String aLastName,

String aFirstName, String aTelephoneNumber, String aDateOfBirth,

String aInsuranceProvider, String aPolicyNumber) {

if (aCustomerId != "" && aCustomerId != null) {

this.setCustomerId(aCustomerId);

}

if (aLastName != "" && aLastName != null) {

this.setLastName(aLastName);

}

if (aFirstName != "" && aFirstName != null) {

this.setFirstName(aFirstName);

}

if (aTelephoneNumber != "" && aTelephoneNumber != null) {

this.setTelephoneNumber(aTelephoneNumber);

}

if (aDateOfBirth != "" && aDateOfBirth != null) {

this.setDateOfBirth(aDateOfBirth);

}

if (aInsuranceProvider != "" && aInsuranceProvider != null) {

this.setInsuranceProvider(aInsuranceProvider);

}

if (aPolicyNumber != "" && aPolicyNumber != null) {

this.setPolicyNumber(aPolicyNumber);

}

}

public String toString() {

String aString = "";

aString = "Customer Id: " + this.getCustomerId() + "\n";

aString = aString + "Last Name: " + this.getLastName() + "\n";

aString = aString + "First Name: " + this.getFirstName() + "\n";

aString = aString + "Telephone Number: " + this.getTelephoneNumber()

+ "\n";

aString = aString + "Date Of Birth: " + this.getDateOfBirth() + "\n";

aString = aString + "Insurance Provider: "

+ this.getInsuranceProvider() + "\n";

aString = aString + "Policy Number: " + this.getPolicyNumber() + "\n";

return aString;

}

}

The above code is formatting the information for when it is called for by the GUI Controller.

**Customers**

The Customers class is a list that holds every Customer in it. When you first initialize the class it calls the DataFileLoader to load all the customers from a text file into the array list.

**Customers**

import java.util.ArrayList;

public class Customers {

private ArrayList<Customer> customerList;

Retrieving Customer information from the file.

public Customers() {

DataFileLoader aFileLoader = new DataFileLoader();

try {

customerList = aFileLoader.getCustomers();

} catch (Exception e) {

e.printStackTrace();

}

}

The following code is saving Customer information to the file.

public void save() {

DataFileLoader aFileLoader = new DataFileLoader();

try {

aFileLoader.saveCustomers(customerList);

} catch (Exception e) {

e.printStackTrace();

}

}

The following code is for retrieving a specific customer’s information.

public Customer getCustomer(String lastName, String firstName) {

for (Customer aCustomer : customerList) {

if (aCustomer.getLastName().equals(lastName)

&& (aCustomer.getFirstName().equals(firstName))) {

return aCustomer;

}

}

return null;

}

public Customer getCustomer(int i) {

return customerList.get(i);

}

public void add(String customerId, String lastName, String firstName,

String telephoneNumber, String dateOfBirth,

String insuranceProvider, String policyNumber) {

this.remove(lastName, firstName);

customerList.add(new Customer(customerId, lastName, firstName,

telephoneNumber, dateOfBirth, insuranceProvider, policyNumber));

}

The following code is to remove a customer .

public void remove(String lastName, String firstName) {

int i = customerList.size() - 1;

while (i >= 0) {

if (customerList.get(i).getLastName().equals(lastName)

&& (customerList.get(i).getFirstName().equals(firstName))) {

customerList.remove(i);

}

i--;

}

}

public boolean contains(String lastName, String firstName) {

boolean isContained = false;

for (Customer aCustomer : customerList) {

if (aCustomer.getLastName().equals(lastName)

&& (aCustomer.getFirstName().equals(firstName))) {

isContained = true;

}

}

return isContained;

}

public int size() {

return customerList.size();

}

public String getCustomerId(int i) {

return customerList.get(i).getCustomerId();

}

public boolean isValid() {

if (this.size() == 0) {

return true;

} else {

return false;

}

}

@Override

public String toString() {

String aString = "";

for (Customer aCustomer : customerList) {

aString = aString + aCustomer.toString() + "\n";

}

return aString;

}

}

**Prescription**

The prescription class is a blueprint for every prescription that is filled by the pharmacy and put into the system. It holds the name of the Physician who prescribed the medication, the telephone number of the physician, the date the prescription was issued by the physician, the medication’s expiration date, the number of refills, and the amount to be refilled. This is all done per customer and added to the customer’s prescription history as an array record.

**Prescription**

public class Prescription {

private String prescriptionId = "N/A";

private String customerId = "N/A";

private String medicationName = "N/A";

private String physicianName = "N/A";

private String physicianPhone = "N/A";

private String dateOfIssue = "N/A";

private String expirationDate = "N/A";

private int numberRefills = 0;

private int unitsPerRefill = 0;

private Boolean generic;

public void setPrescriptionId(String aprescriptionId) {

this.prescriptionId = aprescriptionId;

}

public void setCustomerId(String aCustomerId) {

this.customerId = aCustomerId;

}

public void setMedicationName(String aMedicationName) {

this.medicationName = aMedicationName;

}

public void setPhysicianName(String aPhysicianName) {

this.physicianName = aPhysicianName;

}

public void setPhysicianPhone(String aPhysicianPhone) {

this.physicianPhone = aPhysicianPhone;

}

public void setDateOfIssue(String aDateOfIssue) {

this.dateOfIssue = aDateOfIssue;

}

public void setExpirationDate(String aExpirationDate) {

this.expirationDate = aExpirationDate;

}

public void setNumberRefills(int aNumberRefills) {

this.numberRefills = aNumberRefills;

}

public void setUnitsPerRefill(int aUnitsPerRefill) {

this.unitsPerRefill = aUnitsPerRefill;

}

public void setGeneric(Boolean aGeneric) {

this.generic = aGeneric;

}

public String getPrescriptionId() {

return prescriptionId;

}

public String getCustomerId() {

return customerId;

}

public String getMedicationName() {

return medicationName;

}

public String getPhysicianName() {

return physicianName;

}

public String getPhysicianPhone() {

return physicianPhone;

}

public String getDateOfIssue() {

return dateOfIssue;

}

public String getExpirationDate() {

return expirationDate;

}

public int getNumberRefills() {

return numberRefills;

}

public int getUnitsPerRefill() {

return unitsPerRefill;

}

public Boolean getGeneric() {

return generic;

}

public Prescription() {

This is the empty constructor.

}

public Prescription(String prescriptionId, String customerId,

String medicationName, String physicianName, String physicianPhone,

String dateOfIssue, String expirationDate, int numberRefills,

int unitsPerRefill, boolean generic) {

this.prescriptionId = prescriptionId;

this.customerId = customerId;

this.medicationName = medicationName;

this.physicianName = physicianName;

this.physicianPhone = physicianPhone;

this.dateOfIssue = dateOfIssue;

this.expirationDate = expirationDate;

this.numberRefills = numberRefills;

this.unitsPerRefill = unitsPerRefill;

this.generic = generic;

}

The above code is saving the inputted values of the customer’s prescription.

public String toString() {

String aString = "\n\n";

aString = aString + "Prescription Id: " + this.getPrescriptionId()

+ "\n";

aString = aString + "Customer Id: " + this.getCustomerId() + "\n";

aString = aString + "Medication Name: " + this.getMedicationName()

+ "\n";

aString = aString + "Physician Name: " + this.getPhysicianName() + "\n";

aString = aString + "Physician Phone: " + this.getPhysicianPhone()

+ "\n";

aString = aString + "Date Of Issue: " + this.getDateOfIssue() + "\n";

aString = aString + "Expiration Date: " + this.getExpirationDate()

+ "\n";

aString = aString + "Number Refills: " + this.getNumberRefills() + "\n";

aString = aString + "Units Per Refill: " + this.getNumberRefills()

+ "\n";

aString = aString + "Generic: " + this.getGeneric() + "\n";

return aString;

}

}

The above code is formatting the Prescription record to be stored in the array.

**Prescriptions**

The Prescriptions class is a list that holds every Prescription ever filled by the Pharmacy in it. When you first initialize the class it calls the DataFileLoader to load all the prescription from a text file into the array list.

**Prescriptions**

import java.util.ArrayList;

public class Prescriptions {

private ArrayList<Prescription> prescriptionList = new ArrayList<Prescription>();

public Prescriptions() {

DataFileLoader aFileLoader = new DataFileLoader();

try {

prescriptionList = aFileLoader.getPrescriptions();

} catch (Exception e) {

e.printStackTrace();

}

}

public void save() {

DataFileLoader aFileLoader = new DataFileLoader();

try {

aFileLoader.savePrescriptions(prescriptionList);

} catch (Exception e) {

e.printStackTrace();

}

}

The following code returns customer Id at given array index.

public String getCustomerId(int k) {

return prescriptionList.get(k).getCustomerId();

}

The following code returns Prescription at given array index.

public Prescription getPrescription(int k) {

return prescriptionList.get(k);

}

The following code adds a new prescription. If the prescription exists already, it will remove the old prescription and then adds the new one.

public void add(String prescriptionId, String customerId,

String medicationName, String physicianName, String physicianPhone,

String dateOfIssue, String expirationDate, int numberRefills,

int unitsPerRefill, Boolean generic) {

this.remove(prescriptionId);

prescriptionList.add(new Prescription(prescriptionId, customerId,

medicationName, physicianName, physicianPhone, dateOfIssue,

expirationDate, numberRefills, unitsPerRefill, generic));

}

public void remove(String prescriptionId) {

int i = prescriptionList.size() - 1;

while (i >= 0) {

if (prescriptionList.get(i).getPrescriptionId()

.equals(prescriptionId)) {

prescriptionList.remove(i);

}

i--;

}

}

public boolean contains(String medicationName) {

boolean isContained = false;

for (Prescription aMed : prescriptionList) {

if (aMed.getMedicationName().equals(medicationName)) {

isContained = true;

}

}

return isContained;

}

public int size() {

return prescriptionList.size();

}

public boolean isValid() {

if (this.size() == 0) {

return true;

} else {

return false;

}

}

@Override

public String toString() {

String aString = "";

for (Prescription aMed : prescriptionList) {

aString = aString + aMed.toString();

}

return aString;

}

}

**Drug**

Drug is an interface, meaning it has no variables and just the headers of all it's methods. These methods MUST be called and used by any classes that implement drug.

**Drug**

import java.util.ArrayList;

NOTE: None of these methods have bodies to them.

public interface Drug {

public void setMedicationId(String aMedicationId);

public void setMedicationName(String aMedicationName);

public void setUnitOfMeasurement(String aUnitOfMeasurement);

public void setSideEffectsList(ArrayList<SideEffects> aSideEffect);

public String getMedicationId();

public String getMedicationName();

public String getUnitPrescribed();

public ArrayList<SideEffects> getSideEffectsList();

public String toString();

}

**Medication**

The medication class is a blueprint for every medication that is put into the system. It holds the name of the medication, the units of measurement, possible side effects, and also if there is a generic substitute available for the medication.

**Medication**

import java.util.ArrayList;

public class Medication implements Drug {

private String medicationId;

private String medicationName;

private String unitOfMeasurement;

private ArrayList<SideEffects> sideEffectsList = new ArrayList<SideEffects>();

private Boolean genericsAvailable;

private ArrayList<Generic> genericList = new ArrayList<Generic>();

public void setGenericsAvailable(Boolean aGenericsAvailable) {

this.genericsAvailable = aGenericsAvailable;

}

public Boolean getGenericsAvailable() {

return genericsAvailable;

}

public Medication() {

This is the empty constructor.

}

public Medication(String aMedicationId, String aMedicationName,

String aUnitOfMeasurement, ArrayList<SideEffects> aSideEffectsList,

Boolean aGenericsAvailable) {

super();

this.medicationId = aMedicationId;

this.medicationName = aMedicationName;

this.unitOfMeasurement = aUnitOfMeasurement;

this.sideEffectsList = aSideEffectsList;

this.genericsAvailable = aGenericsAvailable;

}

@Override

public void setMedicationId(String aMedicationId) {

this.medicationId = aMedicationId;

}

@Override

public void setMedicationName(String aMedicationName) {

this.medicationName = aMedicationName;

}

@Override

public void setUnitOfMeasurement(String aUnitOfMeasurement) {

this.unitOfMeasurement = aUnitOfMeasurement;

}

@Override

public void setSideEffectsList(ArrayList<SideEffects> aSideEffectList) {

this.sideEffectsList = aSideEffectList;

}

public void setGeneric(ArrayList<Generic> aGeneric) {

this.genericList = aGeneric;

}

@Override

public String getMedicationId() {

return this.medicationId;

}

@Override

public String getMedicationName() {

return this.medicationName;

}

@Override

public String getUnitPrescribed() {

return this.unitOfMeasurement;

}

@Override

public ArrayList<SideEffects> getSideEffectsList() {

return this.sideEffectsList;

}

public ArrayList<Generic> getGeneric() {

return this.genericList;

}

public void addSideEffects(SideEffects aSideEffect) {

sideEffectsList.add(aSideEffect);

}

public void addGeneric(Generic aGeneric) {

genericList.add(aGeneric);

}

@Override

public String toString() {

String aString = "";

aString = aString + "Medication Id: " + this.getMedicationId() + "\n";

aString = aString + "Medication Name: " + this.getMedicationName()

+ "\n";

aString = aString + "Unit of Measurement: " + this.getUnitPrescribed()

+ "\n";

aString = aString + "Side Effects: \n";

for (int i = 0; i < sideEffectsList.size(); i++) {

aString = aString + "\t" + sideEffectsList.get(i) + "\n";

}

aString = aString + "Generics Available: "

+ this.getGenericsAvailable() + "\n";

if (genericsAvailable) {

aString = aString + "Generic: \n";

aString = aString + genericList.size() + "\n";

for (int i = 0; i < genericList.size(); i++) {

aString = aString + "\t" + genericList.get(i) + "\n";

}

}

aString = aString + "\n";

return aString;

}

}

The above code is formatting the record to be inserted into the Array.

**Medications**

The Medications class is a list that holds every Medication in it. When you first initialize the class it calls the DataFileLoader to load all the medications from a text file into the array list.

**Medications**

import java.util.ArrayList;

public class Medications {

private ArrayList<Medication> medicationList = new ArrayList<Medication>();

This is the default Constructor.

public Medications() {

}

public Medications(Generics generics) {

DataFileLoader aFileLoader = new DataFileLoader();

try {

medicationList = aFileLoader.getMedications(generics);

} catch (Exception e) {

e.printStackTrace();

}

}

The following code adds a new medication to the list. If the medication allready exists the old one will be deleted.

public void add(String medicationId, String medicationName,

String unitOfMeasurement, ArrayList<SideEffects> sideEffectsList,

Boolean genericsAvailable) {

this.remove(medicationName);

medicationList.add(new Medication(medicationId, medicationName,

unitOfMeasurement, sideEffectsList, genericsAvailable));

}

The following code removes medication if the medication name submitted equals another name in the

list.

public void remove(String medicationName) {

int i = medicationList.size() - 1;

while (i >= 0) {

if (medicationList.get(i).getMedicationName()

.equals(medicationName)) {

medicationList.remove(i);

}

i--;

}

}

The following code returns true if medication name is in the list otherwise false.

public boolean contains(String medicationName) {

boolean isContained = false;

for (Medication aMed : medicationList) {

if (aMed.getMedicationName().equals(medicationName)) {

isContained = true;

}

}

return isContained;

}

public ArrayList<Medication> getMedications() {

return medicationList;

}

public int size() {

return medicationList.size();

}

public boolean isValid() {

if (this.size() == 0) {

return true;

} else {

return false;

}

}

@Override

public String toString() {

String aString = "";

for (Medication aMed : medicationList) {

aString = aString + aMed.toString();

}

return aString;

}

}

**Generic**

The Generic class is a blue print for all generics. It holds the medication Id, medication name, unit of measurement and a list of all side effects.

**Generic**

import java.util.ArrayList;

The Generic Class is a single entity of a specific type of Drug.

public class Generic implements Drug {

private String medicationId;

private String medicationName;

private String unitOfMeasurement;

private ArrayList<SideEffects> sideEffectsList = new ArrayList<SideEffects>();

All the @Override methods are methods from Drug that need to be called.

@Override

public void setMedicationId(String aMedicationId) {

this.medicationId = aMedicationId;

}

@Override

public void setMedicationName(String aMedicationName) {

this.medicationName = aMedicationName;

}

@Override

public void setUnitOfMeasurement(String aUnitOfMeasurement) {

this.unitOfMeasurement = aUnitOfMeasurement;

}

@Override

public void setSideEffectsList(ArrayList<SideEffects> aSideEffectsList) {

this.sideEffectsList = aSideEffectsList;

}

@Override

public String getMedicationId() {

return this.medicationId;

}

@Override

public String getMedicationName() {

return this.medicationName;

}

@Override

public String getUnitPrescribed() {

return this.unitOfMeasurement;

}

@Override

public ArrayList<SideEffects> getSideEffectsList() {

return this.sideEffectsList;

}

public void addSideEffects(SideEffects aSideEffect) {

sideEffectsList.add(aSideEffect);

}

public String toString() {

String aString = "";

aString = aString + "Medication Id: " + this.getMedicationId() + "\n";

aString = aString + "Medication Name: " + this.getMedicationName()

+ "\n";

aString = aString + "Unit of Measurement: " + this.getUnitPrescribed()

+ "\n";

aString = aString + "Side Effects: \n";

for (int i = 0; i < sideEffectsList.size(); i++) {

aString = aString + "\t" + sideEffectsList.get(i) + "\n";

}

return aString;

}

}

The above code is formatting the record to be stored in the Array.

**Generics**

The Generics class is a list that holds every Generic in it. When you first initialize the class it calls the DataFileLoader to load all the generics from a text file into the array list.

**Generics**

import java.util.ArrayList;

Generics is basic plural class of Generic, it contains an ArrayList of Generic.

public class Generics {

private ArrayList<Generic> genericList = new ArrayList<Generic>();

The constructor loads pre-existing Generics into the array list from a text file.

public Generics() {

DataFileLoader aFileLoader = new DataFileLoader();

try {

genericList = aFileLoader.getGenerics();

} catch (Exception e) {

TODO Auto-generated catch block

e.printStackTrace();

}

}

public void add(String genericId, String genericName,

String unitOfMeasurement, Boolean genericsAvailable) {

this.remove(genericName);

genericList.add(new Generic());

}

public void remove(String medicationName) {

int i = genericList.size() - 1;

while (i >= 0) {

if (genericList.get(i).getMedicationName().equals(medicationName)) {

genericList.remove(i);

}

i--;

}

}

public String getId(int i) {

return genericList.get(i).getMedicationId();

}

public Generic getGeneric(int i) {

return genericList.get(i);

}

public boolean contains(String medicationName) {

boolean isContained = false;

for (Generic aMed : genericList) {

if (aMed.getMedicationName().equals(medicationName)) {

isContained = true;

}

}

return isContained;

}

public int size() {

return genericList.size();

}

public boolean isValid() {

if (this.size() == 0) {

return true;

} else {

return false;

}

}

@Override

public String toString() {

String aString = "";

for (Generic aMed : genericList) {

aString = aString + aMed.toString();

}

return aString;

}

}

**Side Effects**

SideEffects is an enum which holds a reference to memory. It is implicitly final, because the constants should not be changed.

**SideEffects**

public enum SideEffects {

DROWSINESS("May cause drowsiness"), DIZZINESS("May cause dizziness");

private String description;

private SideEffects(String aDescription) {

this.description = aDescription;

}

The above code is pointing to the proper description of the enum, which is an array to return to the user.

public String toString() {

String aString = "";

aString = description;

return aString;

}

}