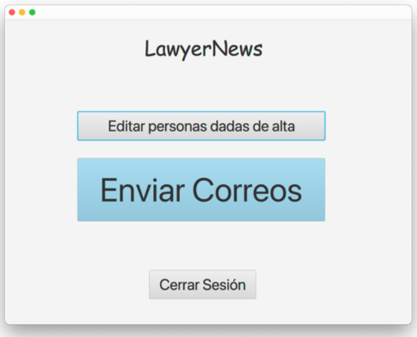
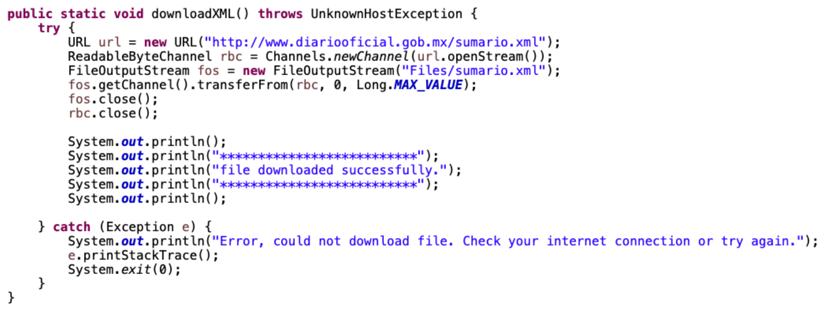
Héctor Pablo González Espinosa A01722968 March 8, 2023

Computer Science Internal Assessment: Criterion C



Techniques Used

The goal of my program, LawyerNews, is to generate personalized e-mails with the sections of the Diario Oficial de la Federación (DOF) that each registered person wishes to receive. My program was written using Java, so I was able to use JavaFX for my GUI. I used CSVs to store the program's users and the people who the program will write e-mails to. To download the DOF off the internet, I used java.io.FileOutputStream, java.net.URL, java.nio.channels.Channels, and java.nio.channels.ReadableByteChannel. Everything was written in Eclipse IDE.



```
public static void downloadXML() throws UnknownHostException {
    try {
        URL url = new URL("http://www.diariooficial.gob.mx/sumario.xml");
        ReadableByteChannel rbc = Channels.newChannel(url.openStream());
        FileOutputStream fos = new FileOutputStream("Files/sumario.xml");
        fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);
        fos.close();
        rbc.close();

        System.out.println();
        System.out.println("**************************");
        System.out.println("file downloaded successfully.");
        System.out.println("**************************");
        System.out.println();

    } catch (Exception e) {
        System.out.println("Error, could not download file. Check your internet connection or try again.");
        e.printStackTrace();
        System.exit(0);
    }
}
```

Table 1a: The code used to download the DOF off the internet

Login Screen and Startup

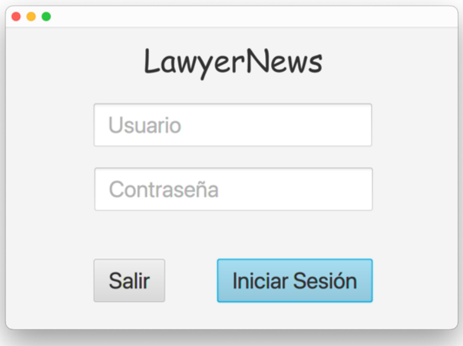
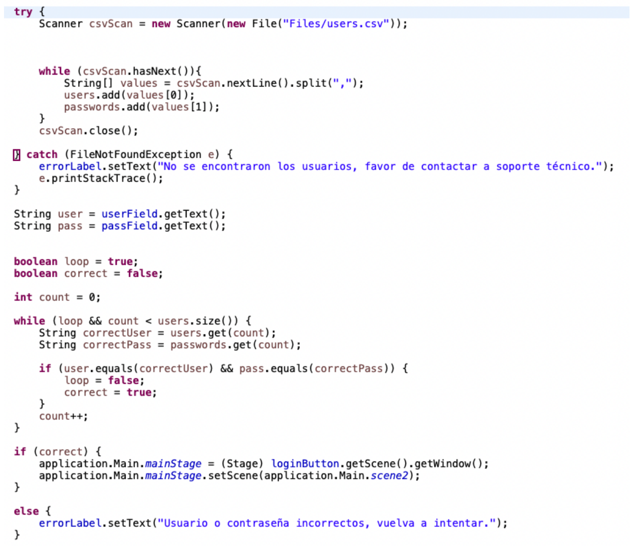


Table 2a: The login screen

My program is protected with a login screen that asks the user for a username and password. The users and passwords are stored in a CSV file. The code cycles through the pairs of users and passwords and compares them with the text in the TextField and PasswordField. If there was no match, the program alerts the user. If the credentials are entered correctly, the program proceeds to the Main window, where the user can choose to edit registered people, send e-mails or logout.



```
try {
    Scanner csvScan = new Scanner(new File("Files/users.csv"));


    while (csvScan.hasNext()){
        String[] values = csvScan.nextLine().split(",");
        users.add(values[0]);
        passwords.add(values[1]);
    }
    csvScan.close();

} catch (FileNotFoundException e) {
    errorLabel.setText("No se encontraron los usuarios, favor de contactar a soporte técnico.");
    e.printStackTrace();
}

String user = userField.getText();
String pass = passField.getText();


boolean loop = true;
boolean correct = false;

int count = 0;

while (loop && count < users.size()) {
    String correctUser = users.get(count);
    String correctPass = passwords.get(count);

    if (user.equals(correctUser) && pass.equals(correctPass)) {
        loop = false;
        correct = true;
    }
    count++;
}

if (correct) {
    application.Main.mainStage = (Stage) loginButton.getScene().getWindow();
    application.Main.mainStage.setScene(application.Main.scene2);
}

else {
    errorLabel.setText("Usuario o contraseña incorrectos, vuelva a intentar.");
}
```

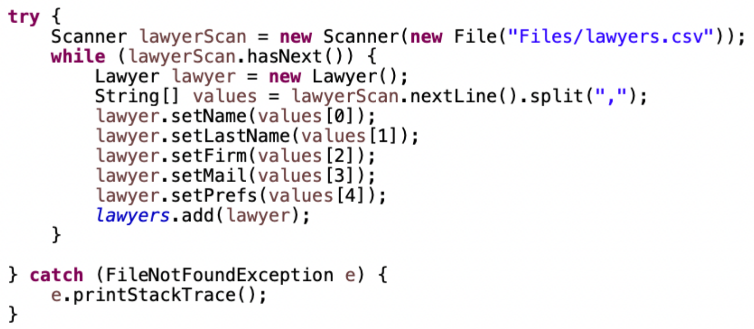
Table 2b: The code used to verify the login

Adding, Editing, and Removing Registered People

The people who will be receiving the e-mails are stored in a CSV file with the following information:

- Name
- Last Name
- Lawyer Firm
- E-Mail
- Preferences

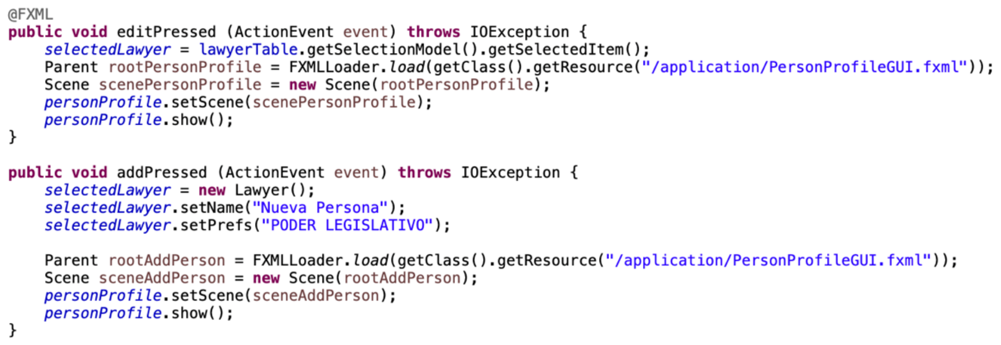
In the LoginController, which is the first Controller to run in the program, a Scanner receives the information from the lawyers.csv file and loads it to memory in the form of an ArrayList of the Lawyer class going by the name of “lawyers.”



```
try {
    Scanner lawyerScan = new Scanner(new File("Files/lawyers.csv"));
    while (lawyerScan.hasNext()) {
        Lawyer lawyer = new Lawyer();
        String[] values = lawyerScan.nextLine().split(",");
        lawyer.setName(values[0]);
        lawyer.setLastName(values[1]);
        lawyer.setFirm(values[2]);
        lawyer.setMail(values[3]);
        lawyer.setPrefs(values[4]);
        lawyers.add(lawyer);
    }

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

Table 3a: Loading lawyers from CSV to memory

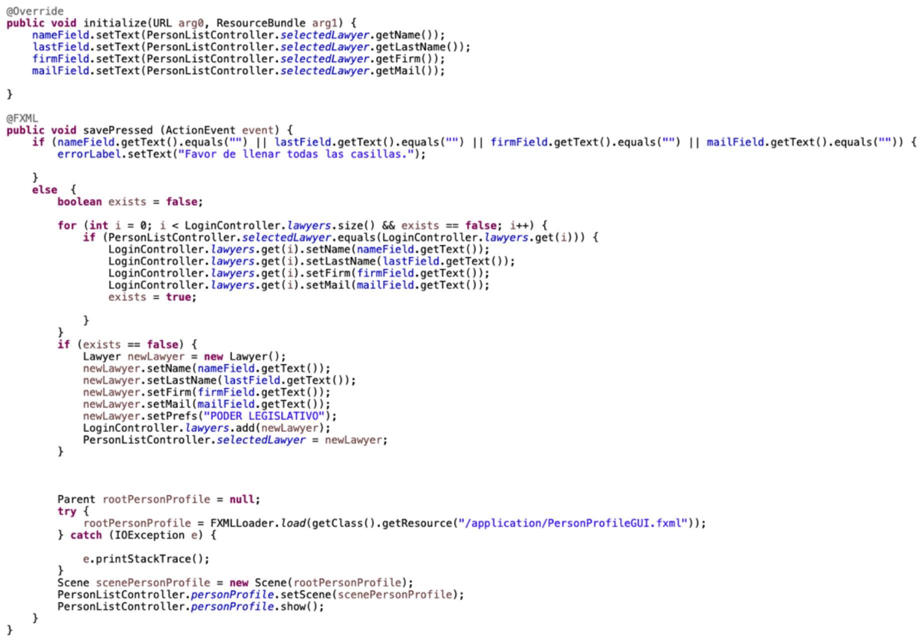
To add or edit the lawyers in the database, the PersonListController displays the lawyers in a TableView. The user can then select one of the lawyers to edit their information or add a new one. Once the user makes a decision, the PersonProfile window opens and gives the user four options: edit information, edit preferences, delete person, or close.



```
@FXML
public void editPressed (ActionEvent event) throws IOException {
    selectedLawyer = lawyerTable.getSelectionModel().getSelectedItem();
    Parent rootPersonProfile = FXMLLoader.load(getClass().getResource("/application/PersonProfileGUI.fxml"));
    Scene scenePersonProfile = new Scene(rootPersonProfile);
    personProfile.setScene(scenePersonProfile);
    personProfile.show();
}

public void addPressed (ActionEvent event) throws IOException {
    selectedLawyer = new Lawyer();
    selectedLawyer.setName("Nueva Persona");
    selectedLawyer.setPrefs("PODER LEGISLATIVO");

    Parent rootAddPerson = FXMLLoader.load(getClass().getResource("/application/PersonProfileGUI.fxml"));
    Scene sceneAddPerson = new Scene(rootAddPerson);
    personProfile.setScene(sceneAddPerson);
    personProfile.show();
}
```

Table 3b: Edit and add buttons

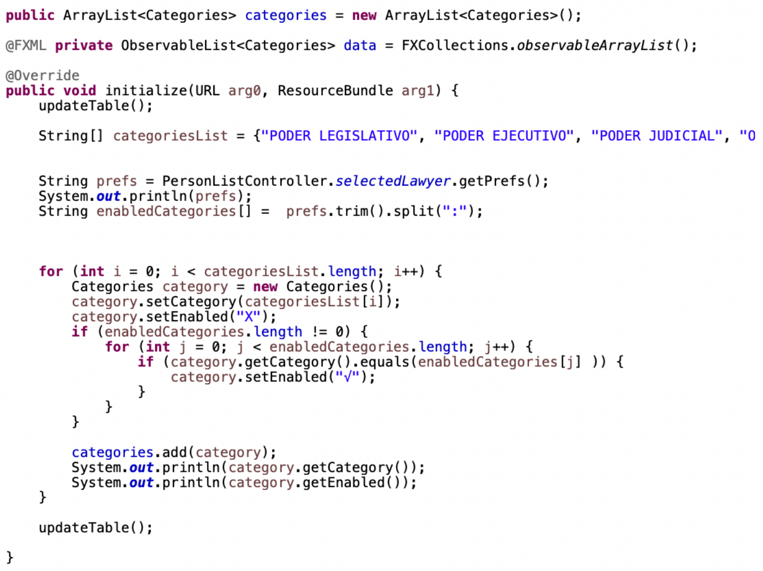
The information can be edited in the Info window. If it’s a new person, the TextFields will be empty. If not, they will be populated with the selected person’s information. All the TextFields must be filled out in order to close the window. Once the close button is pressed, the code checks for empty fields and proceeds if there aren’t any. If an existing person’s information was edited, the code finds the edited lawyer by comparing the selected lawyer with the entries in LoginController.lawyers. When it finds the lawyer, it sets each value to the new values from the TextFields. If it’s a new lawyer, it adds a new lawyer to the ArrayList.



```java
@Override
public void initialize(URL arg0, ResourceBundle arg1) {
    nameField.setText(PersonListController.selectedLawyer.getName());
    lastField.setText(PersonListController.selectedLawyer.getLastName());
    firmField.setText(PersonListController.selectedLawyer.getFirm());
    mailField.setText(PersonListController.selectedLawyer.getMail());

}

@FXML
public void savePressed (ActionEvent event) {
    if (nameField.getText().equals("") || lastField.getText().equals("") || firmField.getText().equals("") || mailField.getText().equals("")) {
        errorLabel.setText("Favor de llenar todas las casillas.");

    }
    else  {
        boolean exists = false;

        for (int i = 0; i < LoginController.lawyers.size() && exists == false; i++) {
            if (PersonListController.selectedLawyer.equals(LoginController.lawyers.get(i))) {
                LoginController.lawyers.get(i).setName(nameField.getText());
                LoginController.lawyers.get(i).setLastName(lastField.getText());
                LoginController.lawyers.get(i).setFirm(firmField.getText());
                LoginController.lawyers.get(i).setMail(mailField.getText());
                exists = true;

            }
        }
        if (exists == false) {
            Lawyer newLawyer = new Lawyer();
            newLawyer.setName(nameField.getText());
            newLawyer.setLastName(lastField.getText());
            newLawyer.setFirm(firmField.getText());
            newLawyer.setMail(mailField.getText());
            newLawyer.setPrefs("PODER LEGISLATIVO");
            LoginController.lawyers.add(newLawyer);
            PersonListController.selectedLawyer = newLawyer;
        }


        Parent rootPersonProfile = null;
        try {
            rootPersonProfile = FXMLLoader.load(getClass().getResource("/application/PersonProfileGUI.fxml"));
        } catch (IOException e) {

            e.printStackTrace();
        }
        Scene scenePersonProfile = new Scene(rootPersonProfile);
        PersonListController.personProfile.setScene(scenePersonProfile);
        PersonListController.personProfile.show();
    }
}
```

Table 3c: InfoController

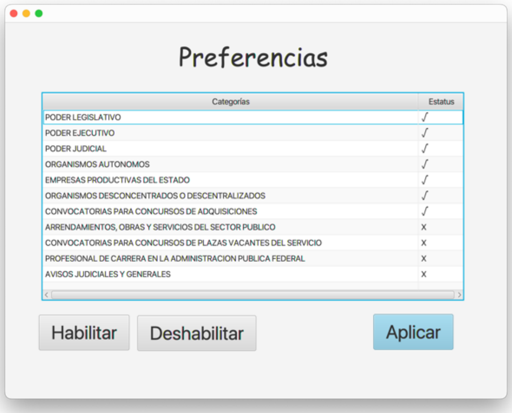
As for the preferences, there are eleven main categories in the DOF. In the Prefs window, the user can enable or disable each of these eleven categories for each person. The person's preferences are stored in the prefs String of the Lawyer class, and ":" is used as a token to separate them. The PrefsController separates them and stores them as an array called enabledCategories. An ArrayList of the Categories class with the eleven categories is created, and for each category it stores a checkmark (√) if the category is present in enabledCategories, and an X if it's not.



```java
public ArrayList<Categories> categories = new ArrayList<Categories>();

@FXML private ObservableList<Categories> data = FXCollections.observableArrayList();

@Override
public void initialize(URL arg0, ResourceBundle arg1) {
    updateTable();

    String[] categoriesList = {"PODER LEGISLATIVO", "PODER EJECUTIVO", "PODER JUDICIAL", "O


    String prefs = PersonListController.selectedLawyer.getPrefs();
    System.out.println(prefs);
    String enabledCategories[] =  prefs.trim().split(":");


    for (int i = 0; i < categoriesList.length; i++) {
        Categories category = new Categories();
        category.setCategory(categoriesList[i]);
        category.setEnabled("X");
        if (enabledCategories.length != 0) {
            for (int j = 0; j < enabledCategories.length; j++) {
                if (category.getCategory().equals(enabledCategories[j] )) {
                    category.setEnabled("√");
                }
            }
        }

        categories.add(category);
        System.out.println(category.getCategory());
        System.out.println(category.getEnabled());
    }

    updateTable();

}
```

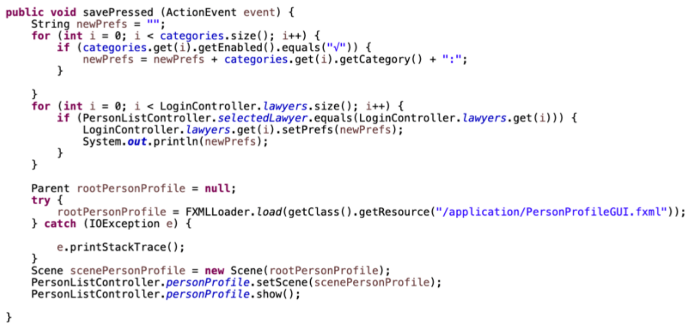
Table 3d: The Initialize method from PrefsController

A TableView then displays the categories with their respective √'s or X's. New users have "PODER LEGISLATIVO" enabled by default. The user can select each category and press the enable or disable buttons, and then apply the changes.



Categorías	Estatus
PODER LEGISLATIVO	√
PODER EJECUTIVO	√
PODER JUDICIAL	√
ORGANISMOS AUTONOMOS	√
EMPRESAS PRODUCTIVAS DEL ESTADO	√
ORGANISMOS DESCONCENTRADOS O DESCENTRALIZADOS	√
CONVOCATORIAS PARA CONCURSOS DE ADQUISICIONES	√
ARRENDAMIENTOS, OBRAS Y SERVICIOS DEL SECTOR PUBLICO	X
CONVOCATORIAS PARA CONCURSOS DE PLAZAS VACANTES DEL SERVICIO	X
PROFESIONAL DE CARRERA EN LA ADMINISTRACION PUBLICA FEDERAL	X
AVISOS JUDICIALES Y GENERALES	X

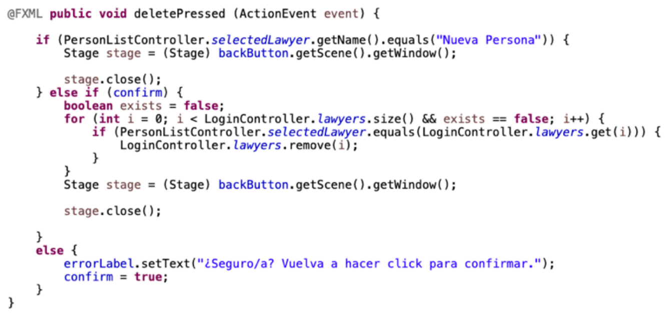
Table 3e: Prefs window



```
public void savePressed (ActionEvent event) {
    String newPrefs = "";
    for (int i = 0; i < categories.size(); i++) {
        if (categories.get(i).getEnabled().equals("√")) {
            newPrefs = newPrefs + categories.get(i).getCategory() + ":";
        }

    }
    for (int i = 0; i < LoginController.lawyers.size(); i++) {
        if (PersonListController.selectedLawyer.equals(LoginController.lawyers.get(i))) {
            LoginController.lawyers.get(i).setPrefs(newPrefs);
            System.out.println(newPrefs);
        }
    }

    Parent rootPersonProfile = null;
    try {
        rootPersonProfile = FXMLLoader.load(getClass().getResource("/application/PersonProfileGUI.fxml"));
    } catch (IOException e) {

        e.printStackTrace();
    }
    Scene scenePersonProfile = new Scene(rootPersonProfile);
    PersonListController.personProfile.setScene(scenePersonProfile);
    PersonListController.personProfile.show();

}
```

Table 3f: Saving changes in prefs

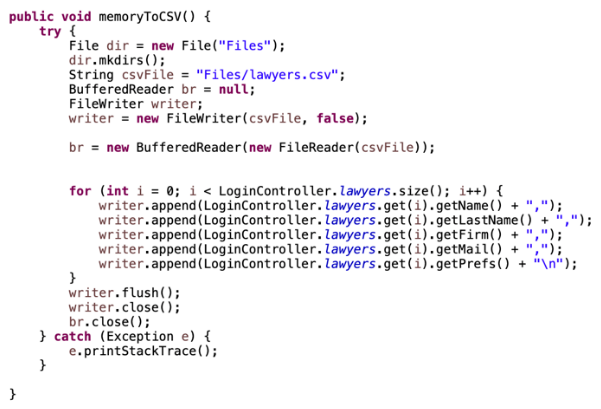
To delete a person, the user must simply press the delete button. A message will alert the user and ask them to press delete again to confirm their action. Afterwards, the changes can be seen once the user presses the "update table" button.



```
@FXML public void deletePressed (ActionEvent event) {

    if (PersonListController.selectedLawyer.getName().equals("Nueva Persona")) {
        Stage stage = (Stage) backButton.getScene().getWindow();

        stage.close();
    } else if (confirm) {
        boolean exists = false;
        for (int i = 0; i < LoginController.lawyers.size() && exists == false; i++) {
            if (PersonListController.selectedLawyer.equals(LoginController.lawyers.get(i))) {
                LoginController.lawyers.remove(i);
            }
        }
        Stage stage = (Stage) backButton.getScene().getWindow();

        stage.close();

    }
    else {
        errorLabel.setText("¿Seguro/a? Vuelva a hacer click para confirmar.");
        confirm = true;
    }
}
```

Table 3g: deletePressed

To save the changes to the CSV, the user must press the “Save Changes” button. The code rewrites lawyers.csv with the entries in LoginControllers.lawyers.

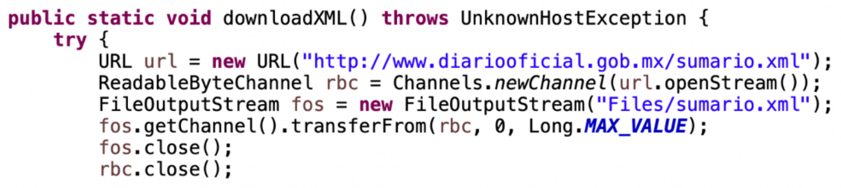


```
public void memoryToCSV() {
    try {
        File dir = new File("Files");
        dir.mkdirs();
        String csvFile = "Files/lawyers.csv";
        BufferedReader br = null;
        FileWriter writer;
        writer = new FileWriter(csvFile, false);

        br = new BufferedReader(new FileReader(csvFile));


        for (int i = 0; i < LoginController.lawyers.size(); i++) {
            writer.append(LoginController.lawyers.get(i).getName() + ",");
            writer.append(LoginController.lawyers.get(i).getLastName() + ",");
            writer.append(LoginController.lawyers.get(i).getFirm() + ",");
            writer.append(LoginController.lawyers.get(i).getMail() + ",");
            writer.append(LoginController.lawyers.get(i).getPrefs() + "\n");
        }
        writer.flush();
        writer.close();
        br.close();
    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

Table 3h: Saving changes to the CSV

Downloading and Sorting the DOF

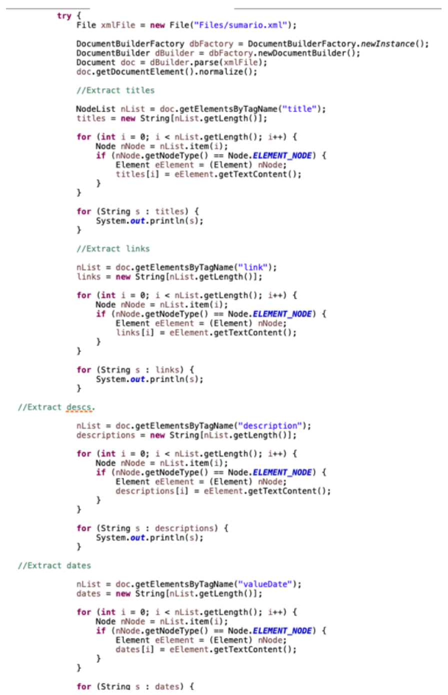
The DOF has an RSS feed, which is downloaded off the internet with the code from XMLTools.downloadXML . I used java.io.FileOutputStream, java.net.URL, java.nio.channels.Channels, and java.nio.channels.ReadableByteChannel to make this possible. The RSS in XML form is obtained from the following URL: http://www.diariooficial.gob.mx/sumario.xml . The resulting file, “sumario.xml”, is saved locally.



```
public static void downloadXML() throws UnknownHostException {
    try {
        URL url = new URL("http://www.diariooficial.gob.mx/sumario.xml");
        ReadableByteChannel rbc = Channels.newChannel(url.openStream());
        FileOutputStream fos = new FileOutputStream("Files/sumario.xml");
        fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);
        fos.close();
        rbc.close();
```

Table 4a: Downloading the CSV

Afterwards, XMLTools.XMLToArticle sorts and loads the articles from sumario.xml in memory in the form of an ArrayList of the Article class that the method returns once run.



```
        try {
            File xmlFile = new File("Files/sumario.xml");

            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(xmlFile);
            doc.getDocumentElement().normalize();

            //Extract titles

            NodeList nList = doc.getElementsByTagName("title");
            titles = new String[nList.getLength()];

            for (int i = 0; i < nList.getLength(); i++) {
                Node nNode = nList.item(i);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    titles[i] = eElement.getTextContent();
                }
            }

            for (String s : titles) {
                System.out.println(s);
            }

            //Extract links

            nList = doc.getElementsByTagName("link");
            links = new String[nList.getLength()];

            for (int i = 0; i < nList.getLength(); i++) {
                Node nNode = nList.item(i);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    links[i] = eElement.getTextContent();
                }
            }

            for (String s : links) {
                System.out.println(s);
            }

  //Extract descs.

            nList = doc.getElementsByTagName("description");
            descriptions = new String[nList.getLength()];

            for (int i = 0; i < nList.getLength(); i++) {
                Node nNode = nList.item(i);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    descriptions[i] = eElement.getTextContent();
                }
            }

            for (String s : descriptions) {
                System.out.println(s);
            }

  //Extract dates

            nList = doc.getElementsByTagName("valueDate");
            dates = new String[nList.getLength()];

            for (int i = 0; i < nList.getLength(); i++) {
                Node nNode = nList.item(i);
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    dates[i] = eElement.getTextContent();
                }
            }

            for (String s : dates) {
```

Table 4b: Sorting the articles

Sending E-Mails

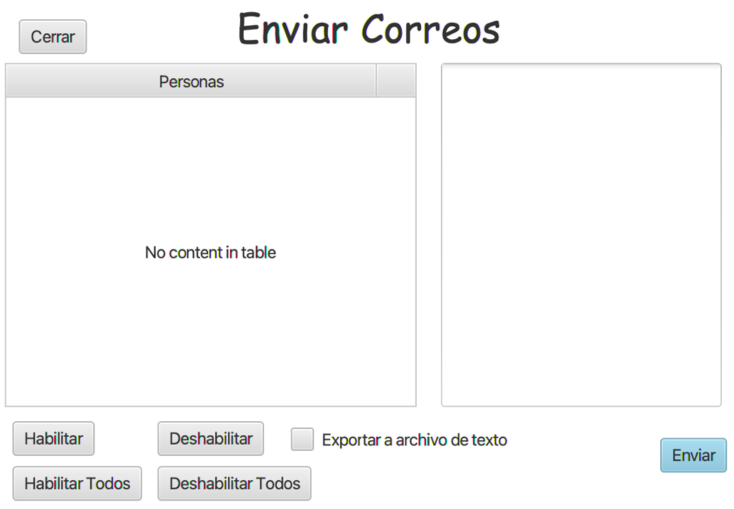
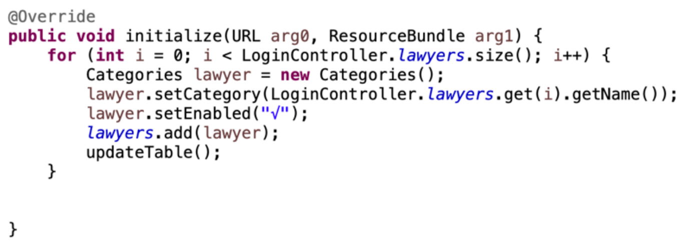


Table 5a: Mail window

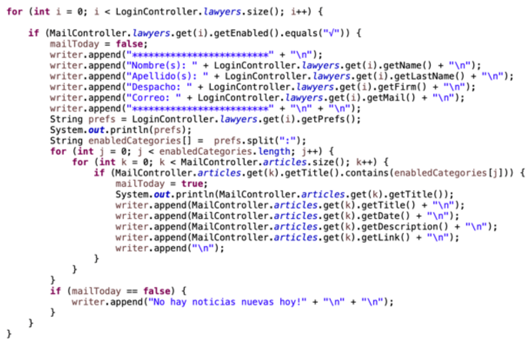
If the user clicks the “Send E-Mails” button in the Main window, the Mail window is opened. All the people registered can be seen in the TableView. By default, all people are enabled. The user can choose which people to enable or disable using the respective buttons. All users can be enabled or disabled at the same time for ease of access. Once the user has decided who to send e-mails to, they can press the “Send” button to proceed. The program will then run XMLTools.DownloadXML and XMLTools.XMLToArticle, which were previously covered. Then, the program runs MailTool.SendMail to send the e-mails using javax.mail.



```java
@Override
public void initialize(URL arg0, ResourceBundle arg1) {
    for (int i = 0; i < LoginController.lawyers.size(); i++) {
        Categories lawyer = new Categories();
        lawyer.setCategory(LoginController.lawyers.get(i).getName());
        lawyer.setEnabled("√");
        lawyers.add(lawyer);
        updateTable();
    }


}
```

Table 5b: The Initialize from MailController

If the user desires it, the e-mails can also be exported to a TXT file. First, it creates a file called “output.txt”. Then, it runs a for loop for the duration of LoginController.lawyers’ size. If the person was enabled in the Mail window with a “√”, the FileWriter by the name of writer will append the person’s name, last name, lawyer firm, and e-mail. Afterwards, the lawyer’s preferences will be split into an array using “:” as the token. Another for loop inside will run for each category in the array. Then, yet another for loop will find the DOF’s articles that correspond to the enabled categories, and append them to the TXT file. If no articles are found, a message saying so will be appended instead.

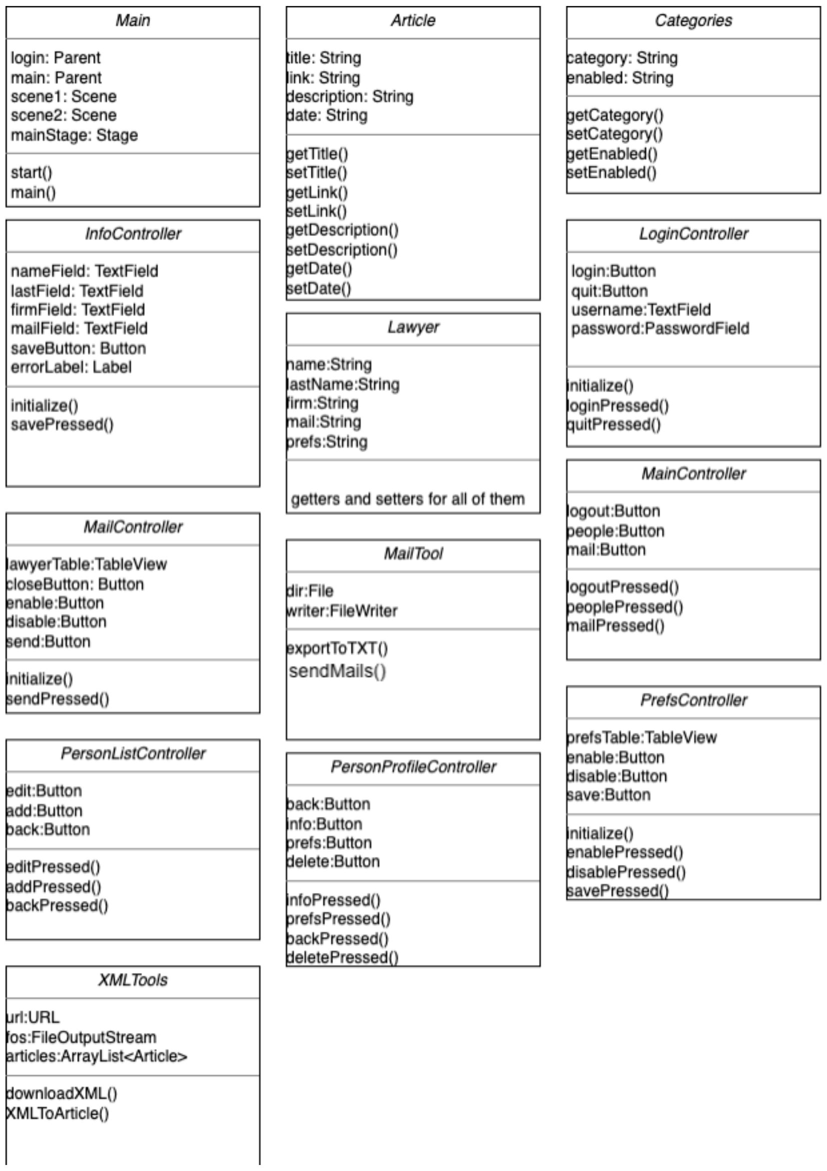


```java
for (int i = 0; i < LoginController.lawyers.size(); i++) {

    if (MailController.lawyers.get(i).getEnabled().equals("√")) {
        mailToday = false;
        writer.append("************************" + "\n");
        writer.append("Nombre(s): " + LoginController.lawyers.get(i).getName() + "\n");
        writer.append("Apellido(s): " + LoginController.lawyers.get(i).getLastName() + "\n");
        writer.append("Despacho: " + LoginController.lawyers.get(i).getFirm() + "\n");
        writer.append("Correo: " + LoginController.lawyers.get(i).getMail() + "\n");
        writer.append("************************" + "\n" + "\n");
        String prefs = LoginController.lawyers.get(i).getPrefs();
        System.out.println(prefs);
        String enabledCategories[] =  prefs.split(":");
        for (int j = 0; j < enabledCategories.length; j++) {
            for (int k = 0; k < MailController.articles.size(); k++) {
                if (MailController.articles.get(k).getTitle().contains(enabledCategories[j])) {
                    mailToday = true;
                    System.out.println(MailController.articles.get(k).getTitle());
                    writer.append(MailController.articles.get(k).getTitle() + "\n");
                    writer.append(MailController.articles.get(k).getDate() + "\n");
                    writer.append(MailController.articles.get(k).getDescription() + "\n");
                    writer.append(MailController.articles.get(k).getLink() + "\n");
                    writer.append("\n");
                }
            }
        }
        if (mailToday == false) {
            writer.append("No hay noticias nuevas hoy!" + "\n" + "\n");
        }
    }
}
```

Table 5c: The triple for loop

After everything is done, the program will let the user know and the emails will be ready in output.txt.

UML Diagram



Word count: 1069 words