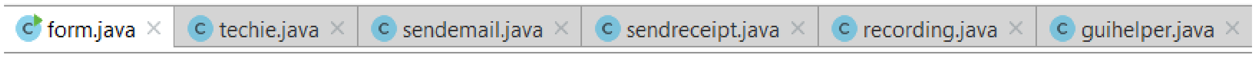
Criterion C: Development

Classes

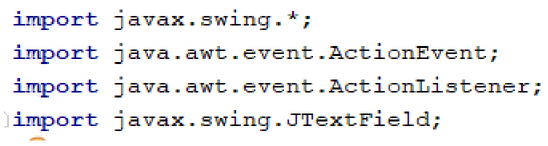


Techniques

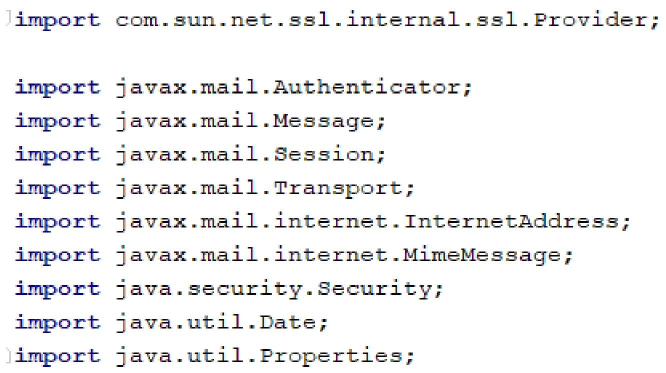
1. Each class extends the main class, form.java
2. Techie.java and recording.java used to record information outputted by the program
3. Sendemail.java and sendreceipt.java allow the main class easy access to the email sending method
4. Guihelper.java helps with using Java swing and creating an interface.

Imports

I needed to import from javax for Java swing, as well as import for my email method.

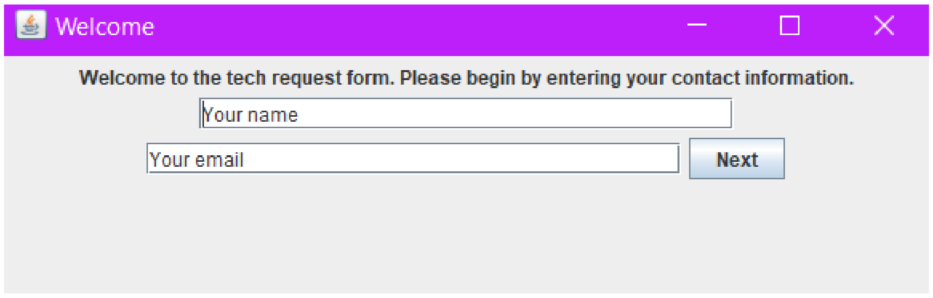


```
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JTextField;
```



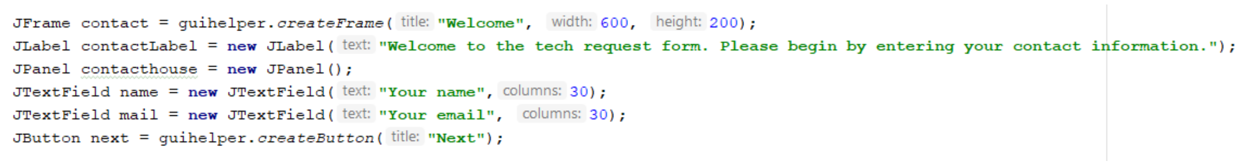
```
import com.sun.net.ssl.internal.ssl.Provider;

import javax.mail.Authenticator;
import javax.mail.Message;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.security.Security;
import java.util.Date;
import java.util.Properties;
```

First JPanel


Welcome
Welcome to the tech request form. Please begin by entering your contact information.
Your name
Your email
Next

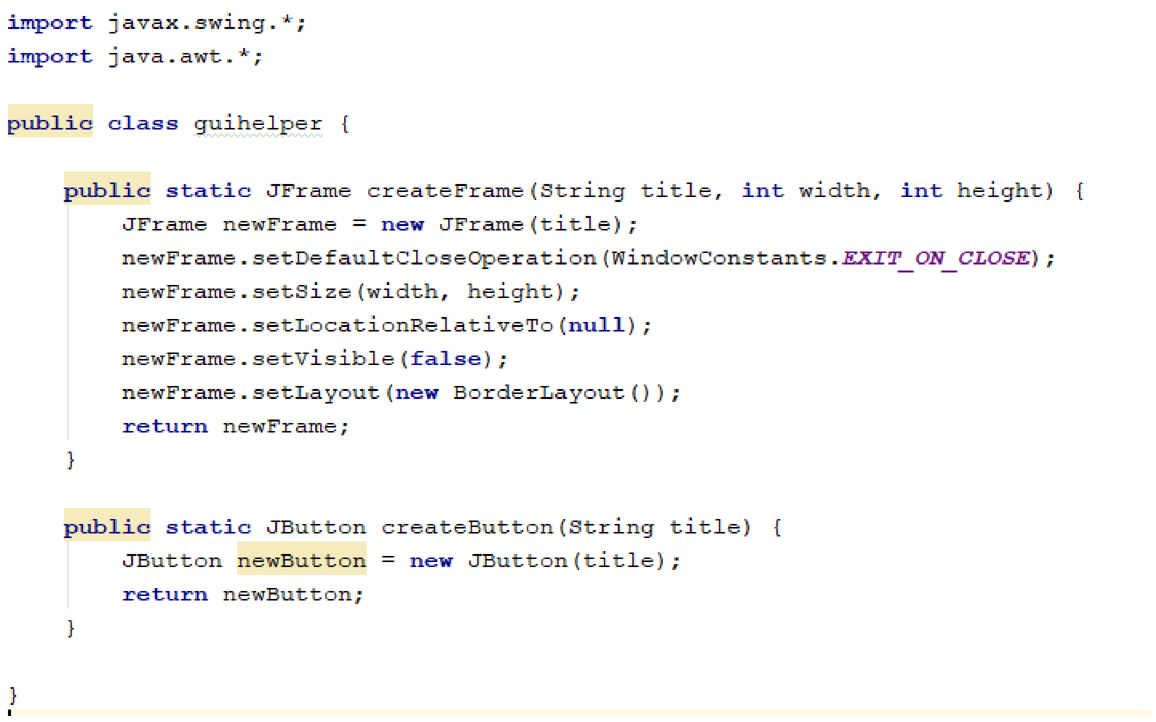
Controls in Java Swing

Creating JPanel, JFrame, JButton, and JTextfield for the first panel in Java swing.



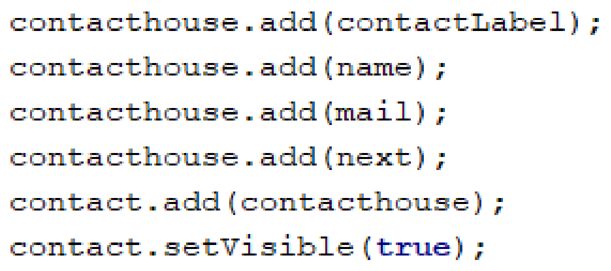
```
JFrame contact = guihelper.createFrame( title: "Welcome",  width: 600,  height: 200);
JLabel contactLabel = new JLabel( text: "Welcome to the tech request form. Please begin by entering your contact information.");
JPanel contacthouse = new JPanel();
JTextField name = new JTextField( text: "Your name",  columns: 30);
JTextField mail = new JTextField( text: "Your email",  columns: 30);
JButton next = guihelper.createButton( title: "Next");
```

Java swing constructors in guihelper class for easier to read, neater code in main class.



```
import javax.swing.*;
import java.awt.*;

public class guihelper {

    public static JFrame createFrame(String title, int width, int height) {
        JFrame newFrame = new JFrame(title);
        newFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        newFrame.setSize(width, height);
        newFrame.setLocationRelativeTo(null);
        newFrame.setVisible(false);
        newFrame.setLayout(new BorderLayout());
        return newFrame;
    }

    public static JButton createButton(String title) {
        JButton newButton = new JButton(title);
        return newButton;
    }

}
```

Adding the appropriate JPanel and JFrame, and then setting the appropriate JFrame visible.

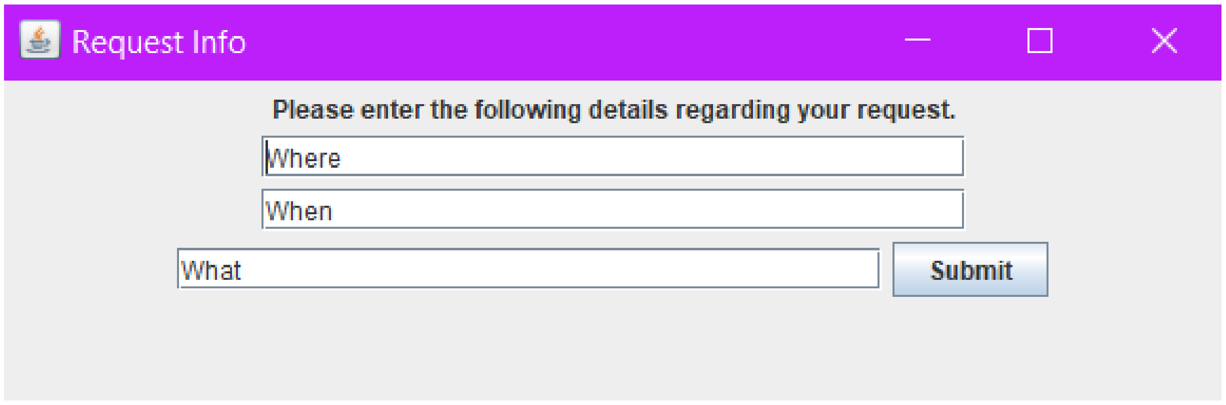


```
contacthouse.add(contactLabel);
contacthouse.add(name);
contacthouse.add(mail);
contacthouse.add(next);
contact.add(contacthouse);
contact.setVisible(true);
```

Implementing an actionlistener for the “next” button, recording data with the recording class, and prompting the next JFrame:

```
next.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        recording.textResponse1 = name.getText();
        recording.textResponse2 = mail.getText();
        System.out.println(recording.textResponse1);
        System.out.println(recording.textResponse2);
        contact.setVisible(false);
        info.setVisible(true);


    }
});
```

A second set of swing components (JLabel, JTextfield, JPanel, and JFrame) is made, just with a different name, and the user is prompted with this window:



Please enter the following details regarding your request.

Where

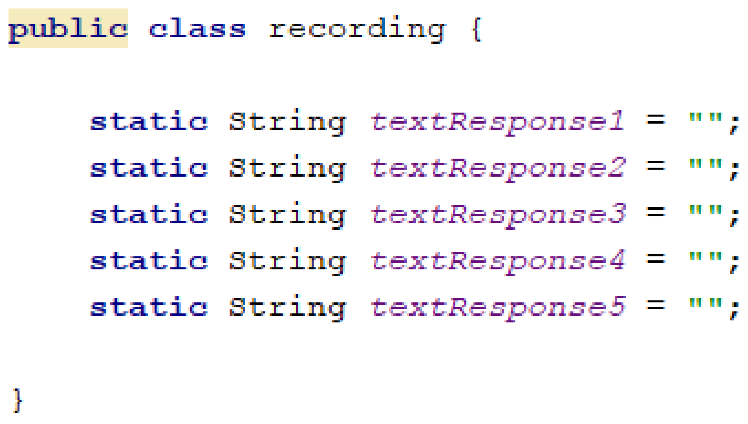
When

What

Actionlistener on the submit button records data into the recording class:

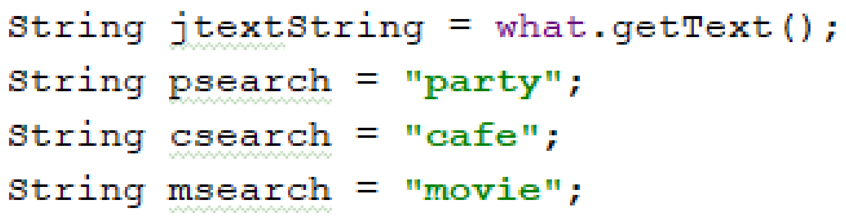
```
submit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        recording.textResponse3 = where.getText();
        recording.textResponse4 = when.getText();
        recording.textResponse5 = what.getText();
        System.out.println(recording.textResponse3);
        System.out.println(recording.textResponse4);
        System.out.println(recording.textResponse5);
        info.setVisible(false);
        ty.setVisible(true);
```

Recording class stores each JTextfield response as a named empty string:



```
public class recording {

    static String textResponse1 = "";
    static String textResponse2 = "";
    static String textResponse3 = "";
    static String textResponse4 = "";
    static String textResponse5 = "";

}
```

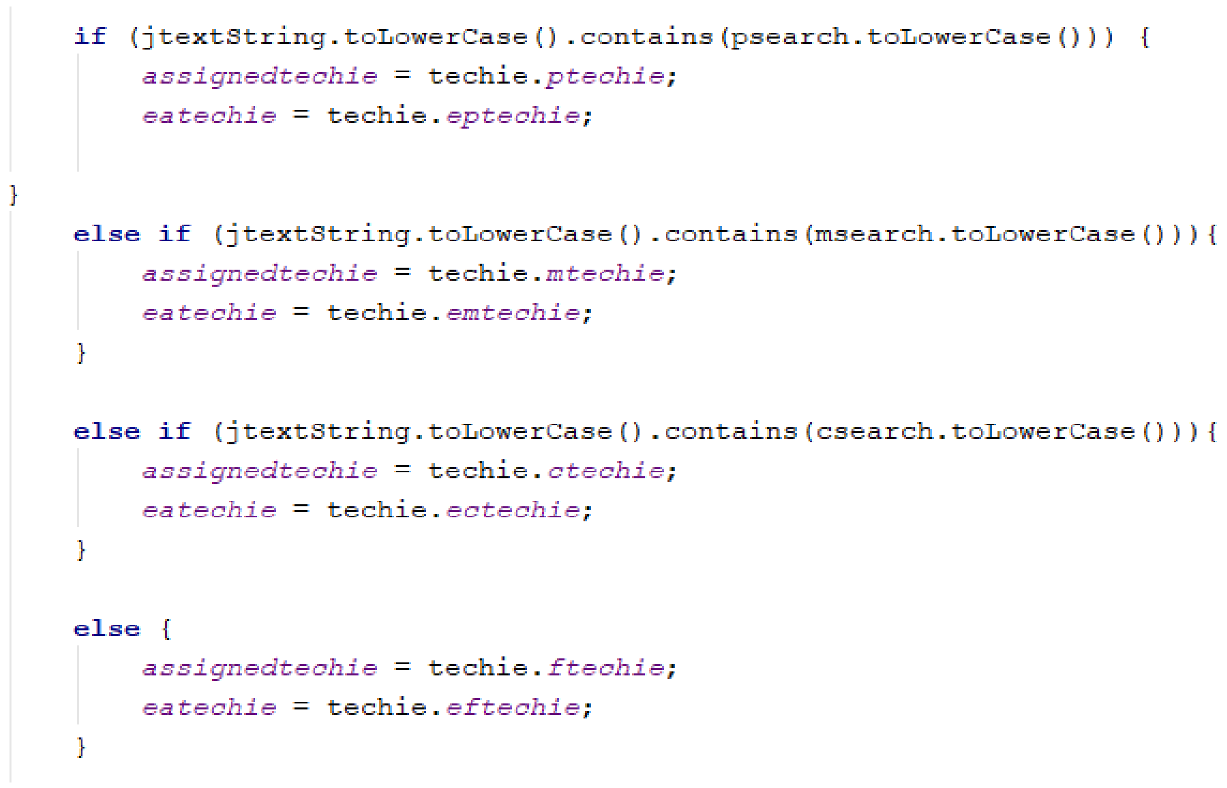
What.getText retrieves text data stored in the “what” Textfield, and saves it as a string. Keywords to be searched are saved as their own strings as well.



```
String jtextString = what.getText();
String psearch = "party";
String csearch = "cafe";
String msearch = "movie";
```

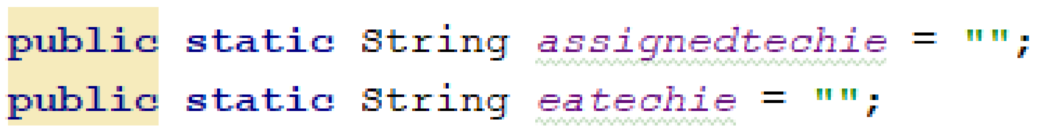
If loop used to assign a techie from the techie class to request.

.contains searches for key words retrieved in **what.getText**. If there are no key words, it is assigned to a techie who does miscellaneous requests.



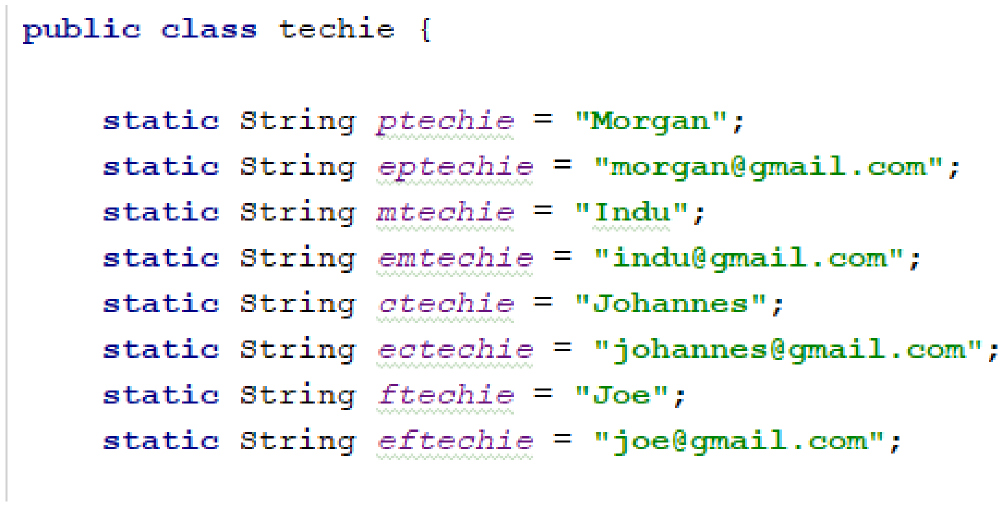
```
    if (jtextString.toLowerCase().contains(psearch.toLowerCase())) {
        assignedtechie = techie.ptechie;
        eatechie = techie.eptechie;

}
    else if (jtextString.toLowerCase().contains(msearch.toLowerCase())){
        assignedtechie = techie.mtechie;
        eatechie = techie.emtechie;
    }

    else if (jtextString.toLowerCase().contains(csearch.toLowerCase())){
        assignedtechie = techie.ctechie;
        eatechie = techie.ectechie;
    }

    else {
        assignedtechie = techie.ftechie;
        eatechie = techie.eftechie;
    }
```

Empty string created earlier in the code to store the name of an assignedtechie and the email of the assigned techie:



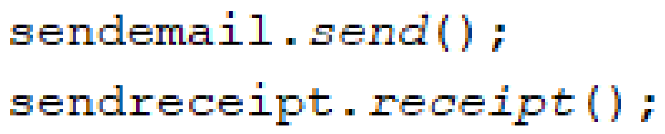
```
public static String assignedtechie = "";
public static String eatechie = "";
```

Techie class used to store the name and email of every techie:



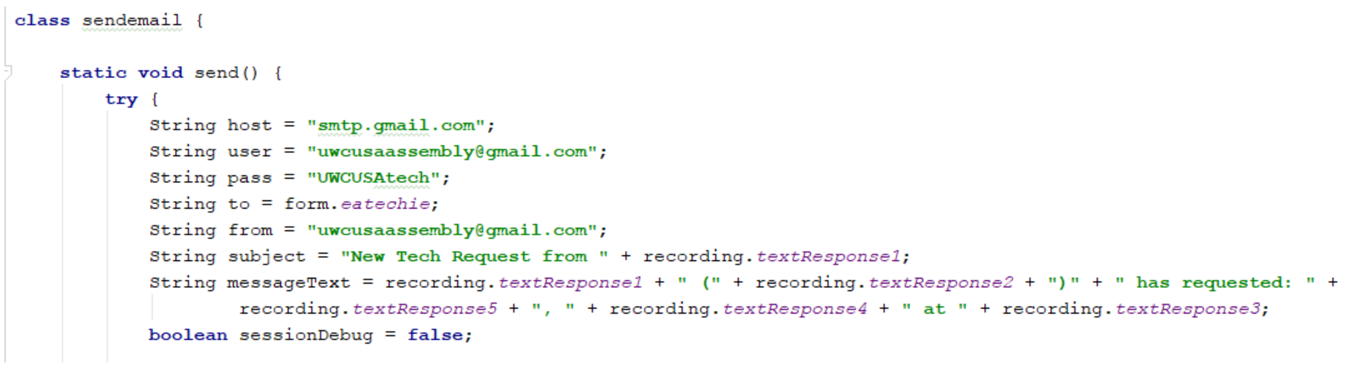
```java
public class techie {

    static String ptechie = "Morgan";
    static String eptechie = "morgan@gmail.com";
    static String mtechie = "Indu";
    static String emtechie = "indu@gmail.com";
    static String ctechie = "Johannes";
    static String ectechie = "johannes@gmail.com";
    static String ftechie = "Joe";
    static String eftechie = "joe@gmail.com";
```

sendemail() and **sendreceipt()** method once a name and email has been stored in the assignedtechie and eatechie string.



```java
sendemail.send();
sendreceipt.receipt();
```

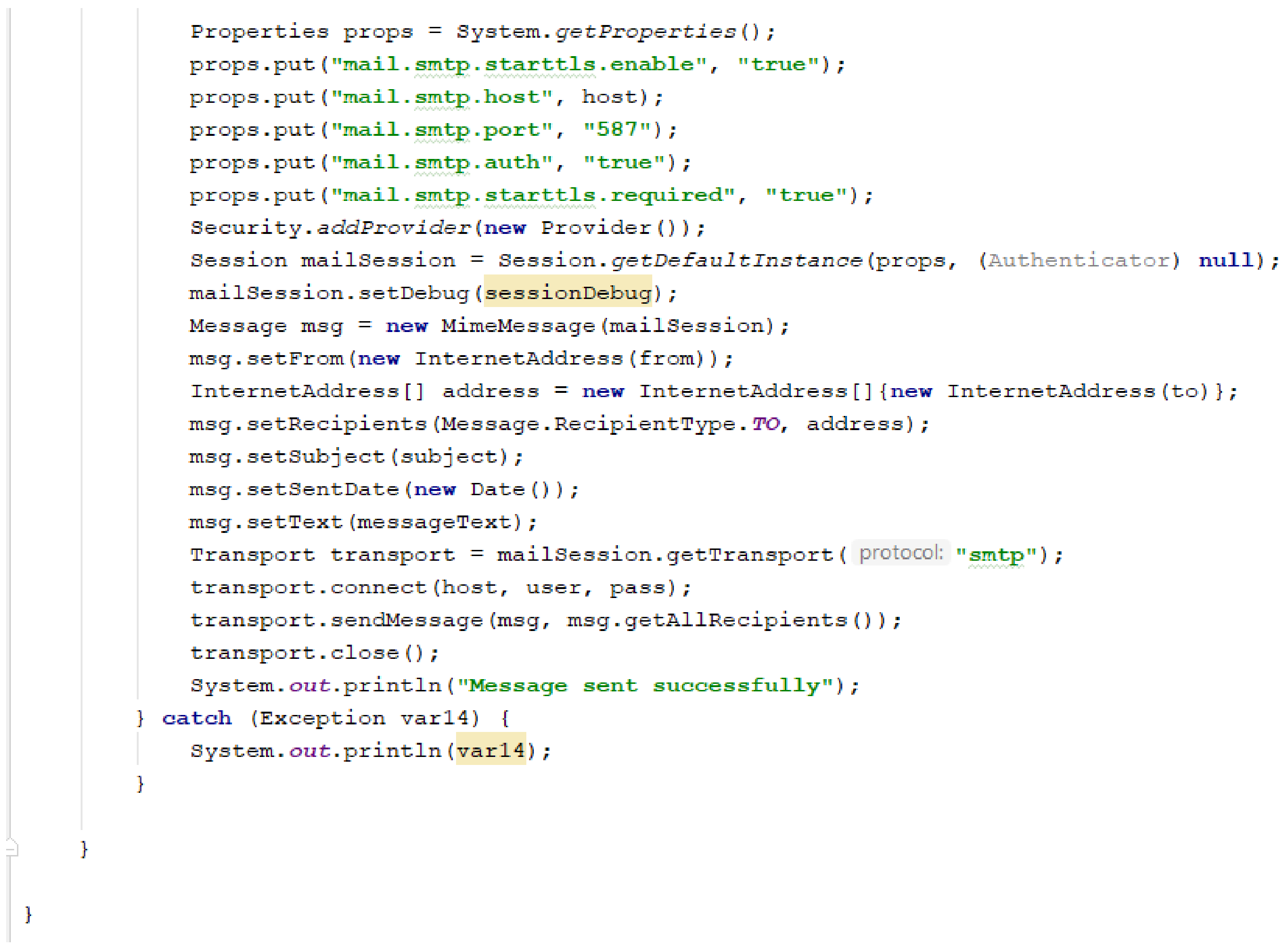
Private class sendemail uses try/catch for Exception var 14, and uses String to store the name of the SMTP server, username, password, and composition of the email:[1]



```java
class sendemail {

    static void send() {
        try {
            String host = "smtp.gmail.com";
            String user = "uwcusaassembly@gmail.com";
            String pass = "UWCUSAtech";
            String to = form.eatechie;
            String from = "uwcusaassembly@gmail.com";
            String subject = "New Tech Request from " + recording.textResponse1;
            String messageText = recording.textResponse1 + " (" + recording.textResponse2 + ")" + " has requested: " +
                    recording.textResponse5 + ", " + recording.textResponse4 + " at " + recording.textResponse3;
            boolean sessionDebug = false;
```

[1] https://www.javatpoint.com/example-of-sending-email-using-java-mail-api

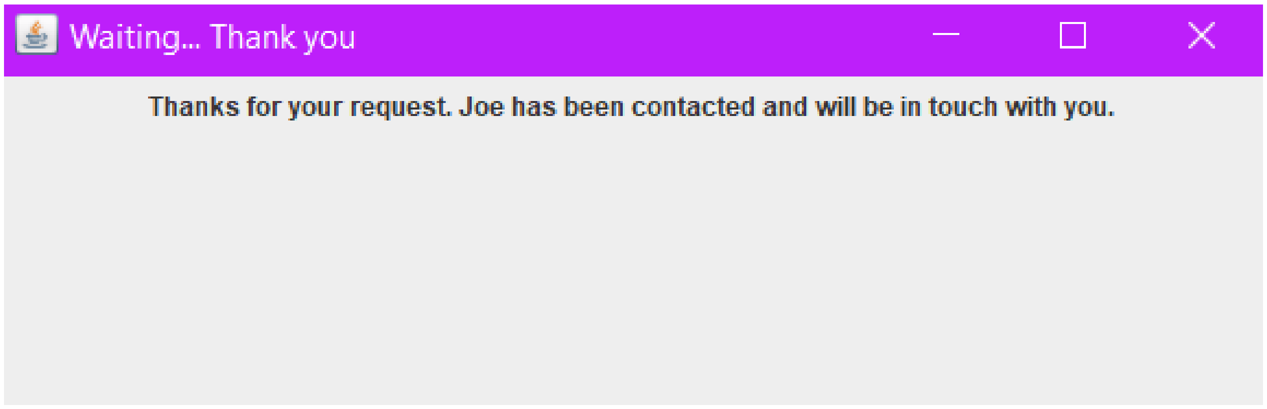
Using imports previously mentioned such as **javax.mail** and **java.util** to connect to host server, authenticate, and send the email using information stored in the Strings.



```
            Properties props = System.getProperties();
            props.put("mail.smtp.starttls.enable", "true");
            props.put("mail.smtp.host", host);
            props.put("mail.smtp.port", "587");
            props.put("mail.smtp.auth", "true");
            props.put("mail.smtp.starttls.required", "true");
            Security.addProvider(new Provider());
            Session mailSession = Session.getDefaultInstance(props, (Authenticator) null);
            mailSession.setDebug(sessionDebug);
            Message msg = new MimeMessage(mailSession);
            msg.setFrom(new InternetAddress(from));
            InternetAddress[] address = new InternetAddress[]{new InternetAddress(to)};
            msg.setRecipients(Message.RecipientType.TO, address);
            msg.setSubject(subject);
            msg.setSentDate(new Date());
            msg.setText(messageText);
            Transport transport = mailSession.getTransport( protocol: "smtp");
            transport.connect(host, user, pass);
            transport.sendMessage(msg, msg.getAllRecipients());
            transport.close();
            System.out.println("Message sent successfully");
        } catch (Exception var14) {
            System.out.println(var14);
        }

    }

}
```

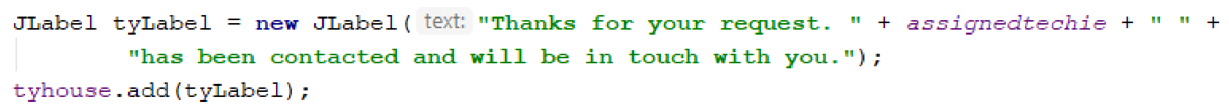
The same is receipted for the **sendreceipt()** method, but with data entered into the String so that the email is sent from **recording.TextResponse2**.

ty.setVisible(true); prompting the final window, with corresponding final JLabel, JFrame, and JPanel:



Thanks for your request. Joe has been contacted and will be in touch with you.

Incorporating **assignedtechie** string for the thank you message. In the above example, the **assignedtechie = ftechie**, who is stored in the techie class as Joe.



```
JLabel tyLabel = new JLabel( text: "Thanks for your request. " + assignedtechie + " " +
        "has been contacted and will be in touch with you.");
tyhouse.add(tyLabel);
```

Clicking the **exit** button ends the program and closes the window.