Technical University of Cluj Napoca

Faculty of Electronics, Telecommunications and Information Technology

Graphic user interface for sending E-mail

Coordinator: Prof. Dr. Ing. Mircea Giurgiu

Coordinator: Asist. Ing. Alexandra Drobuţ

Student: Neamţ Raul

Group: 2031

Java application for sending E-mail

Cluj-Napoca, 2020

Contents

Introduction

Nowadays, everybody on the Internet has an E-mail address which they use for various reasons: communication with other people, creation of accounts on different platforms etc. This project focuses on the communication aspect, making it easier for G-mail users to send an E-mail to another user. Opposed to the classical method of sending a simple text E-mail (Subject + Message), where you have to log in to your account, this application skips this step since the E-mail address and the password must be specified only once, in the code of the application. In addition, the interface is very straightforward, which makes it easier, especially for the new users of E-mail, to communicate online through G-mail.

Theoretical background

Java is a set of computer software and specifications that provides a system for developing application software and deploying it in a cross-platform computing environment. The Java platform is a suite of programs that facilitate developing and running programs written in the Java programming language. A Java platform will include an execution engine, a compiler and a set of libraries.

There were multiple java libraries used in order to make the project work. The first one of them was the java.awt, which deals with the events in the project. The Abstract Window Toolkit (AWT) is Java's original platform-dependent windowing, graphics and user-interface widget toolkit, preceding Swing. The AWT is part of the Java Foundation Classes (JFC) – the standard API for providing a graphical user interface (GUI) for a Java program.

Java.swing is another library that was utilised, due to the fact that it allowed me to design my application with different components like: JLabels, JTextFields and JButtons. Swing is a GUI widget toolkit for Java, which was developed to provide a more sophisticated set of GUI components than earlier Abstract Window Toolkit (AWT). Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent. The term "lightweight" is used to describe such an element, because it does not require allocation of native resources in the operating system's windowing toolkit. Consequently, AWT components are referred to as "heavyweight components".

Finally, the contents of the java.mail library were used to give functionality to the user interface, making it possible to send the E-mail. JavaMail is a Java API used to send and receive

E-mail via SMTP, POP3 and IMAP. JavaMail is built into the Java EE platform, but also provides an optional package for use in Java SE. Since this application is designed to send simple E-mails, the protocol used is SMTP (Simple Mail Transfer Protocol). SMTP is used when E-mail is delivered from an E-mail client to an E-mail server or when E-mail is delivered from one E-mail server to another. SMTP uses parameters like: server name, authentication method, port and connection security. Their role will be defined in the next chapter.

Implementation

In the following chapter I am going to split the source code in small groups in order to present their functionality.

The first group includes the libraries that were used. Their role was specified in the previous chapter.

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.PasswordAuthentication;
import javax.mail.Session;
import javax.mail.Transport;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

Figure 1 Libraries

The next picture shows the definition of the frame variable. Also, it contains public class EventQueue, a platform-independent class that queues events, both from the underlying peer classes and from the trusted application classes. In order to have a functional GUI, any calculation generated by actions of the user must be processed within a different thread, so that the GUI stays responsive. After these calculations, the GUI is updated within the Event Dispatching Thread (EDT). All this is done by EventQueue.invokeLater since it posts the new Runnable event at the end of the Swing event list. Moreover, this event is processed after all previous GUI events are processed. The new Runnable() event is defined by the run() function. This function contains a try-catch block and its role is to make visible the GUI and all its elements. In the case of failure, the catch statement returns the exception that has been caught.

```
private JFrame Email_interface;

public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Send_Email_GUI window = new Send_Email_GUI();
                window.Email_interface.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

Figure 2 Variables declaration and EventQueue class

Another part of the code is represented by the constructor Send_Email_GUI() which contains the initialize method.

```
public Send_Email_GUI() {
    initialize();
}
```

Figure 3 Send_Email_GUI() constructor

The following code block focuses on the initialize function, which is the function that builds both the design and the functionality of the GUI. In the first picture there is presented the initialization of the components: JFrame, JLabel, JTextField and JButton. JFrame's properties are also defined: the size of the window, the close operation and the layout. The properties of JLabel involve just the size. As for the JTextField, there are methods to set the size and the number of columns in the JTextField.

```
private void initialize() {
    //JFrame Email_interface
    Email_interface = new JFrame();
    Email_interface.setBounds(100, 100, 458, 378);
    Email_interface.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Email_interface.getContentPane().setLayout(null);

    //JLabel To
    JLabel To_label = new JLabel("To:");
    To_label.setBounds(62, 43, 46, 13);
    Email_interface.getContentPane().add(To_label);

    //JLabel Subject
    JLabel Subject_label = new JLabel("Subject:");
    Subject_label.setBounds(62, 80, 46, 13);
    Email_interface.getContentPane().add(Subject_label);

    //JLabel Message
    JLabel Message_label = new JLabel("Message:");
    Message_label.setBounds(62, 117, 56, 13);
    Email_interface.getContentPane().add(Message_label);

    //TextField for the address of the e-mail receiver
    JTextField TO_textfield = new JTextField();
    TO_textfield.setBounds(118, 40, 184, 19);
    Email_interface.getContentPane().add(TO_textfield);
    TO_textfield.setColumns(10);

    //TextField for the Subject of the e-mail
    JTextField Subject_Textfield = new JTextField();
    Subject_Textfield.setBounds(118, 77, 184, 19);
    Email_interface.getContentPane().add(Subject_Textfield);
    Subject_Textfield.setColumns(10);

    //TextField for the Message of the e-mail
    JTextField Message_Textfield = new JTextField();
    Message_Textfield.setBounds(118, 114, 184, 130);
    Email_interface.getContentPane().add(Message_Textfield);
    Message_Textfield.setColumns(10);


    //Label for title of the interface
    JLabel title_label = new JLabel("Send Email Interface");
    title_label.setBounds(171, 10, 102, 13);
    Email_interface.getContentPane().add(title_label);

    //Label with authour's name
    JLabel authour_label = new JLabel("Neamt Raul, 2031");
    authour_label.setBounds(328, 318, 106, 13);
    Email_interface.getContentPane().add(authour_label);

    //Button for sending the message
    JButton Send_button = new JButton("SEND");
```

Figure 4 Initialize method: variables initialization

The final part of the code is given by the function which defines the action listener triggered when the “SEND” button is clicked. The two strings that are initialized represent the account details of the sender of the E-mail: username and password. Then we have the Properties variable. Properties is a subclass of Hashtable which is used to maintain lists of values in which the key is a String and the value is also a String. As a result, there are created pairs of parameter-value which are necessary for the SMTP (Simple Mail Transfer Protocol): mail.smtp.host represents the server value (smtp.gmail.com shows that Gmail is used), mail.smtp.port represents the port of the SMTP (here 587 for TLS – Transport Layer Security connection), mail.smtp.auth represents the Authentication method (here the value is “true” which means that an authentication method is used - the password) and mail.smtp.starttls.enable which has the role to provide built-in security (having the value “true” is necessary for this to happen). The Session variable is used for dealing with the interaction between the user and the server and for storing the information of the Properties variable. As a result, a connection is made with the mail server if the PasswordAuthentication(username, password) returns the correct password to the javax.mail.Authenticator.

```
Send_button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        //details of the sender name
        final String username = "sendsmswithjava@gmail.com";
        final String password = "upproject1";

        //properties which allow sending mail: host, port, authentication, tls
        Properties prop = new Properties();
        prop.put("mail.smtp.host", "smtp.gmail.com");
        prop.put("mail.smtp.port", "587");
        prop.put("mail.smtp.auth", "true");
        prop.put("mail.smtp.starttls.enable", "true");

        //creation of session
        Session session = Session.getInstance(prop,
                new javax.mail.Authenticator() {
                    protected PasswordAuthentication getPasswordAuthentication() {
                        return new PasswordAuthentication(username, password);
                    }
                });
```

Figure 5ActionListener variables initialization

In order to test if the message was successfully sent, we use a try-catch statement. Inside the block of the try statement we initialize the value of a Message variable with the default constructor of the MimeMessage class, by using the information of the “session” previously created. (MIME - Multipurpose Internet Mail Extensions, used to determine the file type). Then, using the method setFrom we set the sender of the E-mail to be the address

"smswithjava@gmail.com". Moreover, methods setRecipients, setSubject and setText get the input from the user and set the E-mail address of the recipient, the subject and the content of the message. Finally, the message is sent with Transport.send(message) and the confirmation message will be displayed. In case of failure, the exception is caught and the specific message is displayed.

```
                try {

                    //message variable
                    Message message = new MimeMessage(session);
                    //set sender
                    message.setFrom(new InternetAddress("sendsmswithjava@gmail.com"));
                    //set recipients function
                    message.setRecipients(
                            Message.RecipientType.TO,
                            InternetAddress.parse(TO_textfield.getText())
                    );
                    //set subject and message
                    message.setSubject(Subject_Textfield.getText());
                    message.setText(Message_Textfield.getText());
                    //send message
                    Transport.send(message);
                    //confirmation of sending the message
                    System.out.println("Your e-mail was successfully sent!");

                } catch (MessagingException e) {
                    e.printStackTrace();
                }
            }
        });
        Send_button.setBounds(218, 264, 85, 21);
        Email_interface.getContentPane().add(Send_button);


    }
}
```

Figure 6 Try-catch statement of the ActionListener

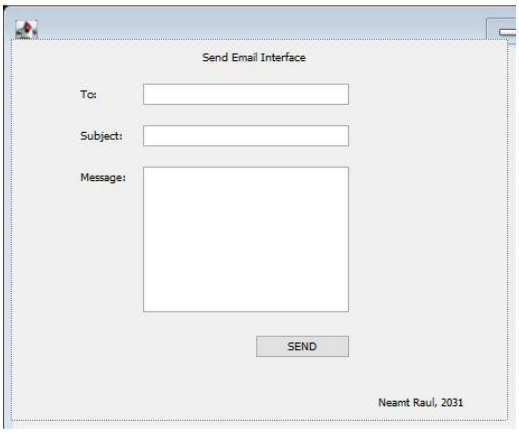

Figure 7 GUI

Conclusions

In conclusion, I think that this project completed the task that it was supposed to do and that its functionality is adequate. This GUI can represent an alternative for beginners who find it hard to use the Gmail application just to send a simple mail. Moreover, this application could be useful for people who communicate by text with Gmail. For example, classmates or co-workers who exchange ideas about different topics via E-mail could highly benefit from using this GUI because they can skip the log in process, which makes the communication faster.

Bibliography:

https://www.hmailserver.com/documentation/v5.6/?page=whatis_pop3imapsmtp

https://www.wikipedia.com

https://stackoverflow.com

https://tutorialspoint.com

https://docs.oracle.com

https://www.programcreek.com/

https://mkyong.com/

https://serversmtp.com/

GitHub link: https://github.com/NeamtRaul/up_project