Tarea Individual 25 - Uso de POP3, lectura de correos

Objetivo

El objetivo de esta actividad es desarrollar una aplicación en Java que permita la conexión a un servidor POP3 (como Gmail), la autenticación mediante un usuario y contraseña, la obtención de los mensajes de correo del buzón de entrada, y la descarga de los mensajes en formato adecuado. Se utilizará la biblioteca `javax.mail` para interactuar con el servidor y realizar estas operaciones.

Descripción de la Tarea

Con base en el código proporcionado, se solicita a los alumnos realizar las siguientes modificaciones y desarrollos:

- **Conexión al servidor POP3:**
  - Utilizar la clase `Store` de la biblioteca `javax.mail` para establecer la conexión con el servidor POP3 especificado (ej. `pop.gmail.com`).
  - Configurar las propiedades de la sesión para utilizar POP3S (POP3 seguro) con SSL, utilizando `mail.store.protocol` y `mail.pop3s.ssl.protocols`.
- **Autenticación:**
  - Implementar la autenticación con el servidor POP3 utilizando las credenciales de un usuario (nombre de usuario y contraseña).
  - Mostrar un mensaje en la consola indicando si la autenticación fue exitosa o fallida.
- **Obtención y almacenamiento de correos:**
  - Conectar al buzón de entrada del servidor y obtener los mensajes utilizando el método `getMessages` de la clase `Folder`.
  - Para cada mensaje, guardar su contenido en un archivo en el sistema de archivos local, con un nombre basado en el asunto del correo. Si el mensaje tiene adjuntos, guardar también esos archivos.
  - Asegurarse de que los mensajes y sus adjuntos sean almacenados correctamente, manejando adecuadamente los tipos de contenido (texto, HTML, etc.).
- **Cierre de sesión y desconexión:**
  - Implementar el cierre de sesión con `close` para el objeto `Folder` y `store.close()` para la desconexión del servidor.
  - Mostrar un mensaje en la consola indicando si la desconexión fue exitosa o no.
- **Uso de TLS 1.2:**
  - props.put("mail.smtp.starttls.enable", "true");
  - props.put("mail.smtp.ssl.protocols", "TLSv1.2");
- **Requisitos adicionales:**
  - Gestionar las excepciones posibles (`MessagingException`, `IOException`) mostrando mensajes claros en la consola.
  - Comentar el código explicando las funcionalidades principales de cada bloque de código.

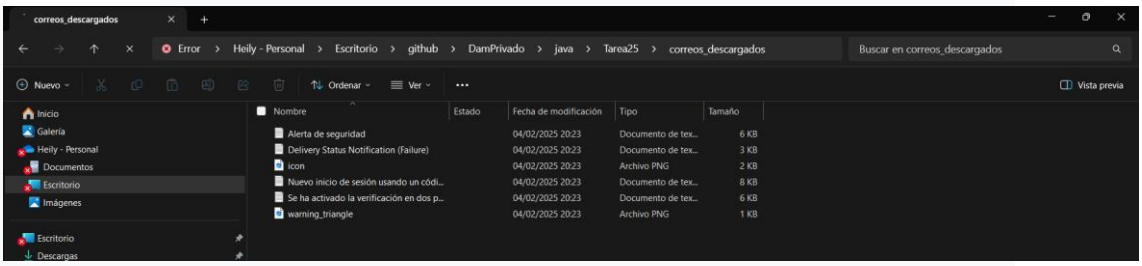
- Asegurarse de que los archivos adjuntos se guarden correctamente en el sistema de archivos, incluyendo la gestión de las extensiones de los archivos.

Entregable

- Capturas de pantalla del código y de la ejecución del programa mostrando:
    - La conexión al servidor POP3 y la autenticación y el cierre de sesión y desconexión del servidor.

```
run:
Conexi�n exitosa al servidor POP3
N�mero de mensajes: 8
Desconexi�n exitosa del servidor POP3
BUILD SUCCESSFUL (total time: 7 seconds)
```

    - La lista de mensajes obtenidos del buzón de entrada y el proceso de almacenamiento de los correos en archivos.



    - Capturas de pantalla del código:

```java
package data;

import javax.mail.*;
import javax.mail.internet.*;
import java.io.*;
import java.util.Properties;

public class Pop3Client {

    private Session session;
    private Store store;
    private String saveDirectory;

    /**
     * Constructor que inicializa el cliente POP3
     * @param saveDirectory Directorio donde se guardarán los correos y adjuntos
     */
    public Pop3Client(String saveDirectory) {
        this.saveDirectory = saveDirectory;
        // Crear el directorio si no existe
        new File(saveDirectory).mkdirs();
    }

    /**
     * Configura y establece la conexión con el servidor POP3
     * @param host Servidor POP3 (ej: pop.gmail.com)
     * @param username Correo electrónico del usuario
     * @param password Contraseña del usuario
     * @throws MessagingException Si hay error en la conexión
     */
    public void connect(String host, String username, String password) throws MessagingException {
        // Configurar propiedades para la conexión segura
        Properties props = new Properties();
        props.put("mail.store.protocol", "pop3s");
        props.put("mail.pop3s.host", host);
        props.put("mail.pop3s.port", "995");
        props.put("mail.pop3s.ssl.protocols", "TLSv1.2");
```

```java
            props.put("mail.smtp.starttls.enable", "true");

            // Crear sesión
            session = Session.getInstance(props);
            // session.setDebug(true); // Descomentar para depuración

            try {
                // Conectar al servidor
                store = session.getStore("pop3s");
                store.connect(host, username, password);
                System.out.println("Conexión exitosa al servidor POP3");
            } catch (MessagingException e) {
                System.err.println("Error en la autenticación: " + e.getMessage());
                throw e;
            }
        }

        /**
         * Descarga todos los mensajes del buzón de entrada
         * @throws MessagingException Si hay error al acceder a los mensajes
         * @throws IOException Si hay error al guardar los archivos
         */
        public void downloadMessages() throws MessagingException, IOException {
            Folder inbox = null;
            try {
                // Abrir carpeta de entrada
                inbox = store.getFolder("INBOX");
                inbox.open(Folder.READ_ONLY);

                // Obtener mensajes
                Message[] messages = inbox.getMessages();
                System.out.println("Número de mensajes: " + messages.length);

                // Procesar cada mensaje
                for (Message message : messages) {
                    processMessage(message);
                }
```

```
        } finally {
            // Cerrar carpeta si está abierta
            if (inbox != null && inbox.isOpen()) {
                inbox.close(false);
            }
        }
    }

    /**
     * Procesa un mensaje individual y sus adjuntos
     * @param message Mensaje a procesar
     * @throws MessagingException Si hay error al procesar el mensaje
     * @throws IOException Si hay error al guardar los archivos
     */
    private void processMessage(Message message) throws MessagingException, IOException {
        // Obtener asunto del mensaje (o usar un valor por defecto si es null)
        String subject = message.getSubject();
        if (subject == null) subject = "Sin_asunto";

        // Sanitizar el nombre del archivo
        String fileName = sanitizeFileName(subject) + ".txt";
        String fullPath = saveDirectory + File.separator + fileName;

        // Guardar contenido del mensaje
        try (PrintWriter writer = new PrintWriter(new FileWriter(fullPath))) {
            writer.println("From: " + InternetAddress.toString(message.getFrom()));
            writer.println("Date: " + message.getSentDate());
            writer.println("Subject: " + subject);
            writer.println("\n--- Contenido ---\n");

            // Procesar el contenido del mensaje
            Object content = message.getContent();
            if (content instanceof Multipart) {
                processMultipart((Multipart) content, writer);
            } else {
                writer.println(content.toString());
            }
```

```
            } else {
                writer.println(content.toString());
            }
        }
    }

    /**
     * Procesa una parte multipart del mensaje
     * @param multipart Parte multipart a procesar
     * @param writer Writer para guardar el contenido del mensaje
     * @throws MessagingException Si hay error al procesar el multipart
     * @throws IOException Si hay error al guardar los archivos
     */
    private void processMultipart(Multipart multipart, PrintWriter writer)
            throws MessagingException, IOException {
        for (int i = 0; i < multipart.getCount(); i++) {
            BodyPart bodyPart = multipart.getBodyPart(i);
            String disposition = bodyPart.getDisposition();

            if (disposition != null && disposition.equalsIgnoreCase(Part.ATTACHMENT)) {
                // Guardar adjunto
                String fileName = bodyPart.getFileName();
                if (fileName != null) {
                    fileName = sanitizeFileName(fileName);
                    String attachmentPath = saveDirectory + File.separator + fileName;

                    try (InputStream is = bodyPart.getInputStream();
                         FileOutputStream fos = new FileOutputStream(attachmentPath)) {
                        byte[] buf = new byte[4096];
                        int bytesRead;
                        while ((bytesRead = is.read(buf)) != -1) {
                            fos.write(buf, 0, bytesRead);
                        }
                    }
                    writer.println("Adjunto guardado: " + fileName);
                }
            } else {
```

```
        } else {
            // Procesar contenido del mensaje
            Object content = bodyPart.getContent();
            if (content instanceof String) {
                writer.println(content);
            } else if (content instanceof Multipart) {
                processMultipart((Multipart) content, writer);
            }
        }
    }
}

/**
 * Sanitiza el nombre del archivo para evitar caracteres inválidos
 * @param fileName Nombre del archivo a sanitizar
 * @return Nombre del archivo sanitizado
 */
private String sanitizeFileName(String fileName) {
    return fileName.replaceAll("[\\\\/:*?\"<>|]", "_");
}

/**
 * Cierra la conexión con el servidor
 */
public void disconnect() {
    try {
        if (store != null && store.isConnected()) {
            store.close();
            System.out.println("Desconexión exitosa del servidor POP3");
        }
    } catch (MessagingException e) {
```

```
            System.err.println("Error al desconectar: " + e.getMessage());
        }
    }

    /**
     * Método principal para probar la funcionalidad
     */
    public static void main(String[] args) {
        Pop3Client client = new Pop3Client("correos_descargados");

        try {
            // Conectar al servidor (ejemplo con Gmail)
            client.connect("pop.gmail.com", "womens.are.women@gmail.com", "ypuz bafp enyr
kchb");

            // Descargar mensajes
            client.downloadMessages();

        } catch (MessagingException e) {
            System.err.println("Error de mensajería: " + e.getMessage());
        } catch (IOException e) {
            System.err.println("Error de E/S: " + e.getMessage());
        } finally {
            // Desconectar
            client.disconnect();
        }
    }
}
```