

Programación de servicios en red II

Programación de servicios y procesos

Contenidos:

- 1) Programación de un cliente SMTP

Programación de un cliente SMTP

El envío y recepción de correos electrónicos desde clases nativas de Java es posible, pero vuelve a ser árido y laborioso. El framework JavaMail dispone de las clases necesarias para poder programar las acciones más habituales en relación con los correos electrónicos, tales como el envío y recepción de mensajes con y sin adjuntos.

Al buscar en Internet tutoriales sobre el envío de correos electrónicos usando Java, existe una alta probabilidad de que cada uno mencione algo llamado Jakarta Mail o Java Mail.

Durante mucho tiempo, Java Enterprise Edition (comúnmente conocido como Java EE), ha sido la plataforma de facto para desarrollar aplicaciones de misión crítica. Recientemente, con el fin de impulsar la creación de aplicaciones nativas de la nube, varios proveedores de software prominentes se unieron para transferir tecnologías Java EE a la Fundación Eclipse, que es una organización sin fines de lucro encargada de administrar las actividades de la comunidad de software de código abierto Eclipse.

En consecuencia, Java EE ha sido renombrado a Jakarta EE.

A pesar del cambio de nombre, todas las clases principales y definiciones de propiedades siguen siendo las mismas tanto para Jakarta Mail como para JavaMail.

Programación de un cliente SMTP

Entonces, Jakarta Mail, o JavaMail como a algunos todavía les gusta llamarlo, es una API para enviar y recibir correos electrónicos a través de SMTP, POP3, así como IMAP y es la opción más popular que también admite la autenticación TLS y SSL. Es independiente de la plataforma, independiente del protocolo e integrado en la plataforma Jakarta EE.

El proceso de envío de correos electrónicos compuestos únicamente por textos consta de los siguientes pasos:

- ✓ Creación de la sesión, indicando la URL del servidor SMTP, el puerto, si utiliza SSL y si se requiere autenticación. Esta configuración de detalles del servidor SMTP se hace utilizando un objeto Java Properties.
- ✓ Creación del mensaje (Objeto **Message**). En este objeto se incluyen tanto la dirección de correo del emisor, la del destinatario, el asunto y el texto del mensaje.
- ✓ Establecimiento de la conexión (creación de objeto de tipo **Transport**), indicando el sistema de transporte.
- ✓ Envío del mensaje.
- ✓ Cierre de la conexión.






Vamos a hacerlo. Crearemos un nuevo proyecto Java para el envío de correos utilizando nuestra cuenta de Google.

Vamos en primer lugar con unos pasos previos y luego continuaremos viendo las clases principales de la API Jakarta Mail.

Programación de un cliente SMTP - Paso 1

Al no tener un servidor de correo propio vamos a utilizar Gmail como servidor de correo. Gmail suele tener activado, y sino debería, la autenticación en 2 pasos y eso impide o dificulta que podemos utilizar directamente nuestro usuario de correo con la password que conocemos. Para poder usar Gmail vamos a generar una contraseña para aplicaciones. Este paso si tuviéramos un servidor de correo propio, nos lo podríamos saltar.

Vamos a Google y a la opción de “Gestionar tu cuenta de Google” y al apartado de “Seguridad”. Dentro vamos al apartado de verificación en 2 pasos:

-  Inicio
-  Información personal
-  Datos y privacidad
-  Seguridad
-  Contactos y compartir

[Revisar actividad de seguridad](#)

Cómo inicias sesión en Google

Asegúrate de poder acceder siempre a tu cuenta de Google manteniendo al día esta información

 Verificación en dos pasos

 Activa desde: 9 nov 2021



Programación de un cliente SMTP - Paso 1

Una vez dentro vamos al apartado de "Contraseñas de aplicación"

Contraseñas de aplicación

Las contraseñas de aplicación no son recomendables y, en la mayoría de los casos, son innecesarias. Para proteger tu cuenta, usa Iniciar sesión con Google para conectar aplicaciones a tu cuenta de Google.

Contraseñas de aplicación

1 contraseña



Programación de un cliente SMTP - Paso 1

Vamos a generar contraseña para aplicaciones y generamos una (que nos tenemos que guardar):

To create a new app specific password, type a name for it below...

App name

Aplicación correo


Crear

Programación de un cliente SMTP - Paso 1

Copiamos en un notepad o similar la contraseña generada

Contraseña de aplicación generada

Tu contraseña de aplicación para el dispositivo



Cómo utilizarla

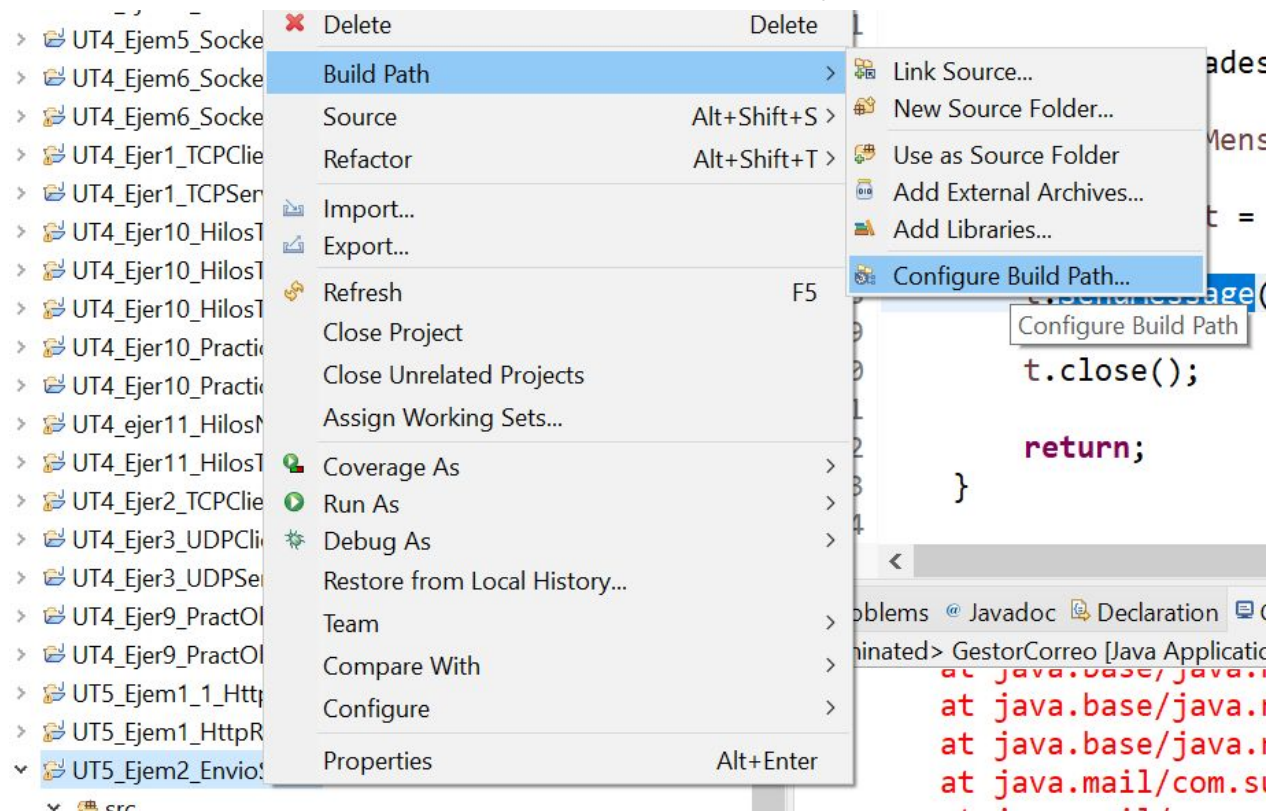
Accede a la sección de configuración de tu cuenta de Google en la aplicación o el dispositivo que estés intentando configurar. Sustituye tu contraseña por la contraseña de 16 caracteres que se muestra arriba.

Al igual que la contraseña normal, esta contraseña de aplicación ofrece acceso completo a tu cuenta de Google. No tendrás que recordarla, así que no la escribas ni la compartas con nadie.

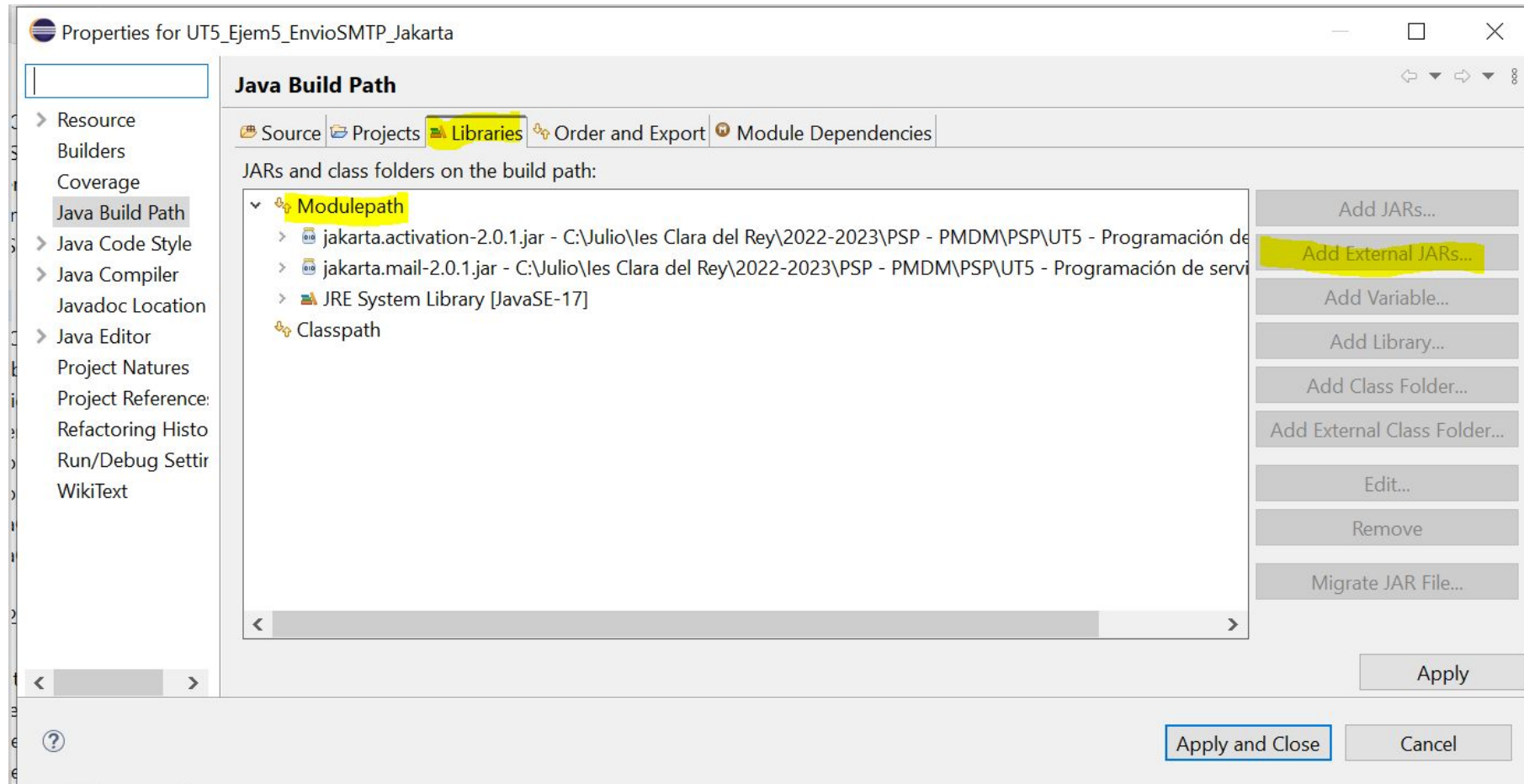
Hecho

Programación de un cliente SMTP - Paso 2

Generamos un proyecto nuevo y utilizaremos el API jakarta.mail. Necesitaremos importar las librerías jakarta.mail y jakarta.activation. Descargamos la librerías (están subidas al aula virtual, pero es muy fácil encontrarlas en internet) y las añadimos al proyecto:



Programación de un cliente SMTP - Paso 2

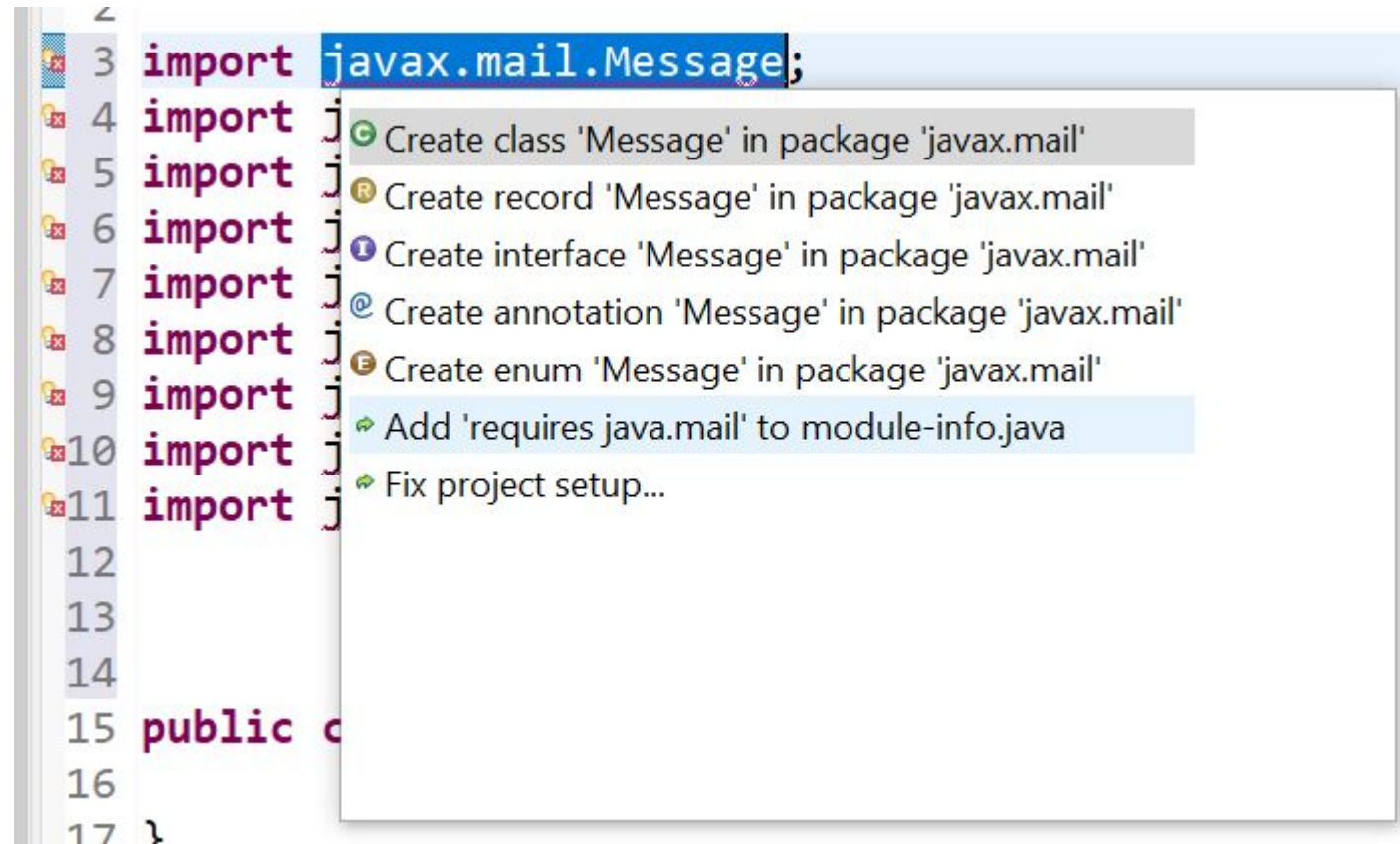


Programación de un cliente SMTP - Paso 2

Crea la clase del proyecto e importa las siguientes librerías:

```
import jakarta.mail.Session;
import jakarta.mail.BodyPart;
import jakarta.mail.Message;
import jakarta.mail.MessagingException;
import jakarta.mail.Multipart;
import jakarta.mail.PasswordAuthentication;
import jakarta.mail.Transport;
import jakarta.mail.internet.InternetAddress;
import jakarta.mail.internet.MimeBodyPart;
import jakarta.mail.internet.MimeMessage;
import jakarta.mail.internet.MimeMultipart;
```

Va a dar error. Necesitas añadirlas al fichero module-info.java (eclipse lo hace por ti con botón derecho y la opción Add...)



Programación de un cliente SMTP - Paso 3

Ya lo tenemos todo, así que vamos a empezar. El esquema de mi proyecto es el siguiente (no tiene que ser igual pero sobre este esquema iré poniendo las explicaciones)

```
public class GestorCorreo {
```

```
    private Properties misPropiedades;
```

```
    private Session miSesion;
```

```
    private MiAutorizacion miAutorizacion;
```

```
    private void setPropiedadesServidorSMTP(String usuario, String password) {
```

```
        Transport conectarServidorSMTP() 
```

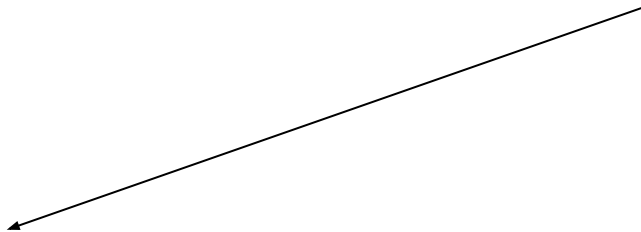
```
    private Message crearNucleoMensaje(String emisor, String destinatario, String asunto) 
```

```
    private Message crearMensajeTexto(String emisor, String destinatario, String asunto, String texto) 
```

```
    private Message crearMensajeTextoConAdjunto(String emisor, String destinatario, String asunto, 
```

```
    private void enviarMensajeTexto(String emisor, String destinatario, String asunto, String texto, 
```

Método para capturar [propiedades](#) que necesitaremos para la conexión y inicializar el objeto sesión



java.util.Properties - Paso 4

La clase **java.util.Properties** es una variante de Map especializada en textos, es decir, un caso particular de Map<String, String>, aunque tiene algunos métodos para facilitar su uso, especialmente escribiendo y leyendo de ficheros de texto.

Métodos más utilizados:

java.util.Properties	
Property()	constructor
Property(Properties defaults)	construye un objeto Properties utilizando otro como defecto; es decir, si el nuevo objeto no tiene la clave que se solicita, se busca en el objeto "defaults".
Object setProperty(String clave, String valor)	asocia el valor a la clave devuelve el antiguo valor asociado a la clave, o NULL si la clave no existía antes
String getProperty(String clave)	devuelve el valor asociado a la clave devuelve NULL si no hay valor asociado
String getProperty(String clave, String X)	devuelve el valor asociado a la clave devuelve X si no hay valor asociado
Set<String> stringPropertyNames()	conjunto de claves que tienen asociado un valor; incluyendo posibles objetos Properties que se usan como respaldo (ver segundo constructor)

java.util.Properties - Paso 4

Para el envío de correo debemos cargar las siguientes propiedades :

- Identificación requerida: clave - mail.smtp.auth / valor - true (ojo, como String, "true")
- Nombre de usuario para acceder al servidor SMTP, (el remitente): clave - mail.smtp.username
- Contraseña de acceso al servidor SMTP (del remitente, el código que hemos copiado de gmail): clave - mail.smtp.password
- Servidor SMTP: clave - mail.smtp.host / valor - smtp.gmail.com
- [Puerto por el que escucha SMTP](#): clave - mail.smtp.port / valor - "587"
- [Propiedades para habilitar la conexión segura](#) (Google no permite hacerlo de otra manera, pero si trabajaras en un entorno "cerrado" y tu aplicación desde la que quieres enviar correos y el servidor de correos están en la misma máquina quizás esto te lo puedas ahorrar)
 - clave - mail.smtp.starttls.enable / valor - true
 - clave - mail.smtp.ssl.trust / valor - smtp.gmail.com

Algo que necesitarás hacer en esta práctica es cargar en un objeto de propiedades, las propiedades de tu equipo (antes de cargar las propiedades descritas anteriormente):

```
misPropiedades = System.getProperties();
```

jakarta.mail.Session - Paso 5

La clase **Session**, que no tiene subclases, es la clase de nivel superior de la API de correo de Jakarta. Es un objeto multiproceso que actúa como fábrica de conexiones para la API de correo de Jakarta. además de recopilar las propiedades y los valores predeterminados de la API de correo, es responsable de los ajustes de configuración y la autenticación.

Para **obtener** el objeto **Session**, puede llamar a cualquiera de los dos métodos siguientes:

- `getDefaultInstance()`, que devuelve la sesión predeterminada
- `getInstance()`, que devuelve una nueva sesión. El parámetro que se le pasa debe recoger al menos las siguientes propiedades: **servidor smtp, puerto, conexión segura y si requiere identificación**. Además hay un par de formas de autenticarse y una ellas, la que vamos a usar, requiere meter el usuario y password como propiedades también.

Para obtener el objeto Sesión que utilizaremos vamos a utilizar método `getInstance()`. Para utilizar este método necesitaremos el objeto de propiedades definido antes y necesitaremos definir un objeto que extienda de la clase `Authenticator` y en la que sobre escribamos un método para que utilice el usuario y password de nuestro correo (SIEMPRE que hablemos de password será del código para contraseñas que hemos generado en Google)

jakarta.mail.Session - Paso 5

```
public class MiAutorizacion extends Authenticator{

    private String user;
    private String pass;

    public MiAutorizacion (String user, String pass) {
        this.user = user;
        this.pass = pass;
    }

    @Override
    public PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(user, pass);
    }
}
```


jakarta.mail.Transport - Paso 6

La clase **Transport** es una clase abstracta que utiliza el protocolo SMTP para enviar y transportar mensajes de correo electrónico. Hereda de la clase Service, la cual proporciona funcionalidades comunes a todos los servicios de mensajería, tales como conexión, desconexión, transporte y almacenamiento.

Cuando tengamos construido el mensaje hay que enviarlo a través del **protocolo SMTP**. Para ello necesitamos un objeto de tipo Transport como hemos visto antes. Este objeto nos lo devuelve nuestro de Sesión con el método getTransport en el que simplemente debemos indicar el protocolo utilizado ("smtp").

Este objeto establecerá la conexión para el "transporte de los mensajes" y tiene los métodos para el envío:

- Transport.send(message): le pasamos o bien un único argumento (el objeto Message en cuestión).
- Transport.sendMessage: tiene la ventaja de que aprovechamos la misma conexión al servidor para enviar muchos mensajes, mientras que con send() se establece una conexión diferente para cada uno de ellos. Para utilizar esta opción la conexión debe hacerse no al configurar la sesión sino al configurar el objeto Transport

Utilizaremos el Transport.send(message)

La clase Address es una *clase abstracta* que modela las direcciones (direcciones To y From) en un mensaje de correo electrónico; Sus subclases soportan las implementaciones reales. Por lo general, su `subclase InternetAddress`, que denota una dirección de correo electrónico de Internet, es la que mas se usa.

jakarta.mail.Message - Paso 7

La clase Message es una clase abstracta que modela un mensaje de correo electrónico. Sus subclases soportan las implementaciones reales. Por lo general, su subclase **MimeMessage** (jakarta.mail.internet.MimeMessage) se utiliza para preparar los detalles del mensaje de correo electrónico que se enviará. Un MimeMessage es un mensaje de correo electrónico que utiliza el estilo de formato MIME (Multipurpose Internet Mail Extension)

Estos son algunos de los métodos más utilizados de la clase MimeMessage:

Método	Descripción
setFrom(Address addresses)	Se utiliza para establecer el campo de encabezado "De".
setRecipients(Message.RecipientType type, String addresses)	Se utiliza para establecer el tipo de destinatario indicado en las direcciones proporcionadas. Los posibles tipos de direcciones definidos son "Para" (Message.RecipientType.TO), "CC" (Message.RecipientType.CC) y "CCO" (Message.RecipientType.BCC).
setSubject(String subject)	Se utiliza para establecer el campo de encabezado del asunto del correo electrónico.
setText(String text)	Se utiliza para establecer la cadena proporcionada como contenido del correo electrónico, utilizando el tipo MIME de "texto / sin formato".
setContent(Object message, String contentType)	Se utiliza para establecer el contenido del correo electrónico y se puede utilizar con un tipo MIME que no sea "text/html".

jakarta.mail.Message - Paso 7

Para crear un mensaje sencillo, sin adjunto, necesitaremos utilizar los siguientes métodos:

- Crear el objeto mensaje construyendo este mensaje con nuestro objeto de sesión.
- Crear un objeto de tipo `InternetAddress` con la dirección de correo del emisor.
- Añadir al emisor del mensaje (`setFrom`).
- Crear un objeto de tipo `InternetAddress` con la dirección de correo del receptor
- Añadir al receptor (`.addRecipient`). Puedes añadir varios si quieres y si están en el para o en copia.
- Añadir el contenido del mensaje (`setContent`). En este método le indicamos que el contenido es texto/html ("`text/html`")

Una vez enviado y como hemos dicho antes, con nuestro objeto `Transport` y el método `send` enviamos el mensaje. Después de enviarlo cerramos (`close`) el objeto `transport`.

Main - Paso 8

Y acabamos en el main, donde pedimos por consola los datos del correo

```
Console ×
<terminated> GestorCorreo (1) [Java Application] C:\Program Files\Java\jdk-17.0.1\bin\javaw.exe (28 nov 2024, 20:10:07 – 20:10:44) [pid: 23
Introduce tu destinatario
jhornosb@gmail.com
Introduce el asunto
HOLA CLASE DA2D1E
Introduce el texto del mensaje
Bienvenidos al maravilloso mundo del protocolo SMTP
Desea incluir un adjunto
NO
nov 28, 2024 8:10:42 P. M. jakarta.mail.Session loadResource
WARNING: expected resource not found: /META-INF/javamail.default.address.map
Mensaje enviado
```

Programación de un cliente SMTP: Adjuntos

Además de los pasos mencionados anteriormente, estos son los diferentes pasos involucrados en el uso de la API de correo de Jakarta para enviar archivos adjuntos de correo electrónico:

- ✓ Creamos un objeto de tipo Message con los datos de “Cabecera” (emisor, receptor y asunto)
- ✓ Creamos un objeto de tipo MimeBodyPart donde con el método setContent indicamos el cuerpo del mensaje y su formato ("text/html")
- ✓ Creamos otro objeto de tipo MimeBodyPart y con el método attachFile adjuntamos el adjunto (valga la redundancia).
 - Este fichero adjunto lo hemos tenido que “meter” previamente en un objeto de tipo File que es la abstracción de java para ficheros,
- ✓ Creamos una instancia del objeto MimeMultipart que se utilizará como envoltorio o wrapper para las distintas partes MimeBodyPart. Un MultiPart actúa como un contenedor que guarda varias partes individuales, y viene con métodos para obtener y configurar sus diversas subpartes (addBodyPart).
- ✓ Incluimos el objeto Multipart en el mensaje a enviar (nuevamente setContent).

