



# Programación de Servicios y Procesos

## 5.1 Correo

---



I.E.S.  
Doctor Balmis

Apuntes de PSP ([https://psp2dam.github.io/psp\\_sources/es/](https://psp2dam.github.io/psp_sources/es/)) creados por Vicente Martínez bajo licencia  
CC BY-NC-SA 4.0  (<http://creativecommons.org/licenses/by-nc-sa/4.0/deed.es>)

## 5.1 Correo

- 5.1.1 Jakarta Mail
  - Uso de la librería
- 5.1.2 Clases principales de la API Jakarta Mail
  - jakarta.mail.Session
  - jakarta.mail.Message
  - jakarta.mail.Address
  - jakarta.mail.Authenticator
  - jakarta.mail.Transport
- 5.1.3 Enviar correos electrónicos básicos en Jakarta Mail
  - Preparación de la sesión
  - Composición del mensaje (texto sin formato)
  - Enviar mensaje
- 5.1.4 Enviar mensajes HTML
- 5.1.5 Enviar correo electrónico con archivos adjuntos
- 5.1.6 Enviar correos electrónicos HTML con imágenes
  - Insertando imágenes CID (Content-ID)
  - Insertando imágenes en línea (Base64)
  - Linked images
- 5.1.7 Lectura de emails con Jakarta Mail

### 5.1.1 Jakarta Mail

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.



#### Jakarta vs Java Mail

Para evitar confusiones, es importante tener en cuenta que JavaMail es solo el nombre anterior de Jakarta Mail y los dos representan el mismo software.

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.

También puede encontrar Jakarta Mail como un paquete opcional para su uso con la plataforma Java SE.

## Uso de la librería

**JakartaMail API** (<https://jakarta.ee/specifications/mail/2.1/jakarta-mail-spec-2.1>) es la especificación de la API de correo de Jakarta. La implementación de referencia de esta especificación se puede encontrar en el repositorio de GitHub **Jakarta Mail Specification** (<https://jakartaee.github.io/mail-api/>) .

### JavaMail API

Como su nombre indica, JavaMail API es una API, no una implementación. Por lo tanto, no puede usarlo directamente en su proyecto. En cambio, debe usar una implementación de la API de correo de Jakarta, como la implementación de referencia de la especificación de la API de correo de Jakarta.

También puede encontrar el archivo jakarta.mail-api-X.Y.Z.jar en el repositorio de Maven y agregarlo con dependencias **Maven** :

```
<dependencies>
  <dependency>
    <groupId>org.eclipse.angus</groupId>
    <artifactId>angus-mail</artifactId>
    <version>2.0.2</version>
    <type>jar</type>
  </dependency>
  <dependency>
    <groupId>jakarta.mail</groupId>
    <artifactId>jakarta.mail-api</artifactId>
    <version>2.1.2</version>
    <type>jar</type>
  </dependency>
</dependencies>
```

xml

### Implementaciones de Jakarta Mail

En el ejemplo anterior hemos usado la implementación de Jakarta AngusMail, pero se pueden usar otras como la propia de Oracle, simplemente cambiando la dependencia.

```
<dependency>
  <groupId>com.sun.mail</groupId>
  <artifactId>jakarta.mail</artifactId>
  <version>2.0.1</version>
</dependency>
```

xml

Al ser una implementación del API propuesta por Jakarta, el código debería funcionar sin cambios.

## 5.1.2 Clases principales de la API Jakarta Mail

La API de correo de Jakarta tiene una amplia gama de clases e interfaces que se pueden usar para enviar, leer y realizar otras acciones con mensajes de correo electrónico, al igual que en un sistema de correo típico.

Aunque hay varios paquetes en el Proyecto de Correo de Jakarta, dos de los más utilizados son `jakarta.mail` y `jakarta.mail.internet`.

El paquete `jakarta.mail` proporciona clases que modelan un sistema de correo y el paquete `jakarta.mail.internet` proporciona clases que se centran en los sistemas de correo de Internet.

A continuación tenemos una descripción de las clases principales de cada uno de los paquetes:

## `jakarta.mail.Session`

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

## `jakarta.mail.Message`

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)** definido en el RFC822.

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

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

## `jakarta.mail.Address`

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.Authenticator`

La clase `Authenticator` es una *clase abstracta* que se utiliza para obtener autenticación para acceder a los recursos del servidor de correo, a menudo requiriendo la información del usuario. Por lo general, su `subclase PasswordAuthentication` es la que mas se usa.

## jakarta.mail.Transport

La clase `Transport` es una *clase abstracta* que utiliza el `protocolo SMTP` para enviar y transportar mensajes de correo electrónico.

### 5.1.3 Enviar correos electrónicos básicos en Jakarta Mail

Básicamente, estos son los pasos para enviar un email utilizando la API de correo de Jakarta:

1. Configurar detalles del servidor SMTP utilizando un **objeto Java Properties**. Puede obtener detalles del servidor SMTP de su proveedor de servicios de correo electrónico.
2. Crear un objeto `Session` llamando al método `getInstance()`. Luego, pasar el `nombre de usuario y contraseña de la cuenta` a `PasswordAuthentication`. Al crear el objeto de sesión, siempre debe registrar el `Authenticator` con la sesión.
3. Una vez creado el objeto `Session`, el siguiente paso es crear el email que se enviará. Para ello, empezaremos pasando el objeto de sesión creado al constructor de clase `MimeMessage`.
4. A continuación, después de crear el objeto de mensaje, establecer los campos De, Para y Asunto del email.
5. Utilizar el método `setText()` para establecer el contenido del mensaje de correo electrónico.
6. Utilizar el objeto `Transport` para enviar el correo.
7. Agregar excepciones para recuperar los detalles de cualquier posible error al enviar el mensaje.

#### Preparación de la sesión

El primer paso es obtener el objeto de sesión de correo. La clase `Session` es una clase singleton. Por lo tanto, no se puede crear directamente una instancia de la misma. Debemos llamar a uno de los métodos estáticos sobrecargados, generalmente

`getInstance()` .



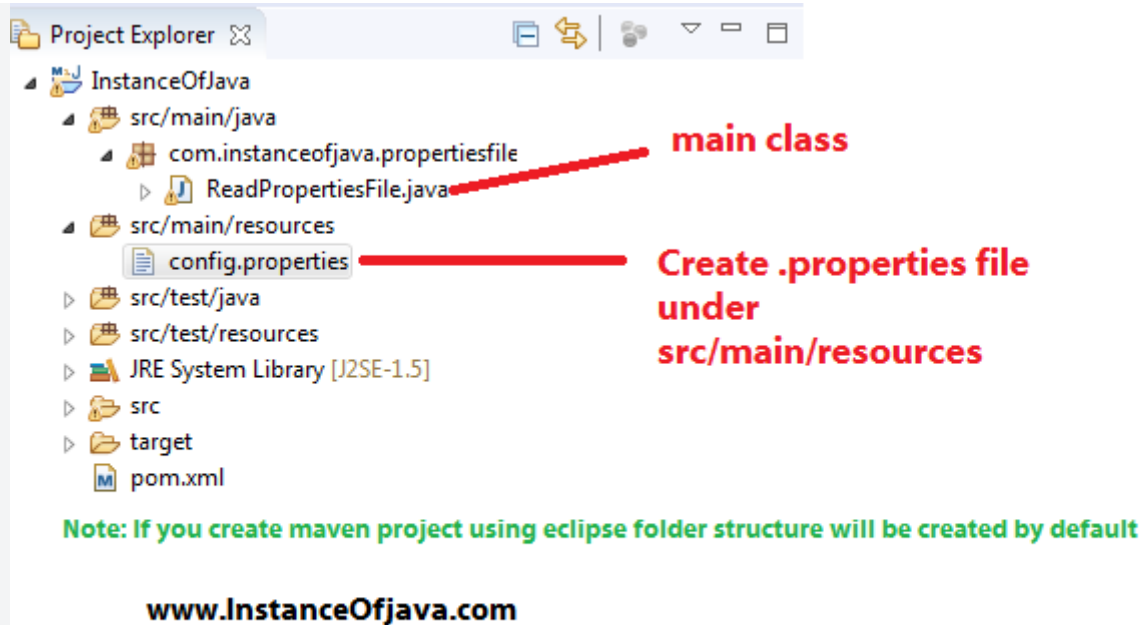
#### Archivos/Objetos Properties

Un archivo de propiedades es un archivo de texto que contiene pares clave-valor para los valores de configuración de un proyecto.

Se pueden crear sobre la marcha como en los siguientes ejemplos, pero también se puede leer desde un archivo en el proyecto (esta es la forma preferida).

Aquí puedes encontrar algunos enlaces para echar un vistazo a cómo usar y acceder a estos archivos

- [Java Properties Files y como usarlos - Arquitectura Java](https://www.arquitecturajava.com/java-properties-files-y-como-usarlos/) (<https://www.arquitecturajava.com/java-properties-files-y-como-usarlos/>)
- [Getting started with Java properties - Baeldung](https://www.baeldung.com/java-properties) (<https://www.baeldung.com/java-properties>)
- [Properties class in Java - javaTpoint](https://www.javatpoint.com/properties-class-in-java) (<https://www.javatpoint.com/properties-class-in-java>)



En la imagen anterior puedes ver dónde colocar el archivo de propiedades en un proyecto 'Maven'.

```

1 // Prepare SMTP configuration into a Property object
2 final Properties prop = new Properties();
3 prop.put("mail.smtp.username", "usuario@gmail.com");
4 prop.put("mail.smtp.password", "passwordEmail");
5 prop.put("mail.smtp.host", "smtp.gmail.com");
6 prop.put("mail.smtp.port", "587");
7 prop.put("mail.smtp.auth", "true");
8 prop.put("mail.smtp.starttls.enable", "true"); // TLS
9
10 // Create the Session with the user credentials
11 Session mailSession = Session.getInstance(prop, new jakarta.mail.Authenticator() {
12     @Override
13     protected PasswordAuthentication getPasswordAuthentication() {
14         return new PasswordAuthentication(prop.getProperty("mail.smtp.username"),
15             prop.getProperty("mail.smtp.password"));
16     }
17 });

```

java

En el código anterior, acabamos de crear el objeto Session con propiedades y el objeto Authenticator.

Las propiedades son las siguientes.

- mail.smtp.username – Nombre de usuario para acceder al servidor SMTP
- mail.smtp.password – Contraseña de acceso al servidor SMTP
- mail.smtp.host – Host del servidor SMTP
- mail.smtp.port – Puerto
- mail.smtp.auth – Se requiere autenticación.
- mail.smtp.starttls.enable – TLS habilitado o no.

El Authenticator es una clase abstracta. Su objeto se crea proporcionando una implementación anónima del método `getPasswordAuthentication()`. La clase `PasswordAuthentication` se utiliza como marcador para almacenar credenciales de

usuario.

Este es un código de ejemplo para leer las propiedades de un archivo en el proyecto.

```

1      private void loadSMTPConfiguration() {
2          try ( InputStream input = this.getClass().getResourceAsStream("/") + propertiesFile)) {
3
4              smtpConfiguration = new Properties();
5              // Load a properties file
6              smtpConfiguration.load(input);
7
8              // get the property value and print it out
9              smtpConfiguration.forEach((key, value) -> System.out.println("Key : " + key + ", Value : " + value));
10
11         } catch (IOException ex) {
12             System.err.println("Cant open properties file: " + ex.getLocalizedMessage());
13             ex.printStackTrace();
14         }
15     }

```

El contenido del archivo de propiedades podría ser así

```

1      # Data to send emails from a GMAIL account
2      mail.from=cuenta@iesdoctorbalmis.com ó cuenta@gmail.com
3      mail.username=cuenta@iesdoctorbalmis.com
4      mail.password=***contraseña de la cuenta habilitando apps poco seguras o contraseña de app con 2FA***
5
6      mail.smtp.host=smtp.gmail.com
7      mail.smtp.port=587
8      mail.smtp.auth=true
9      mail.smtp.starttls.enable=true

```

## Composición del mensaje (texto sin formato)

A continuación, redactaremos el mensaje de correo electrónico. La clase `jakarta.mail.Message` representa un mensaje en Java mail API. Dado que es una clase abstracta, usaremos su clase de implementación concreta `jakarta.mail.internet.MimeMessage`. Java Mail API permite enviar correo en texto plano o en contenido HTML. Comencemos enviando un mensaje de texto sin formato.

```

1      // Prepare the MimeMessage
2      Message message = new MimeMessage(mailSession);
3      // Set From and subject email properties
4      message.setFrom(new InternetAddress("no-reply@gmail.com"));
5      message.setSubject("Sending Mail with pure Java Mail API ");
6
7      // Set to, cc & bcc recipients
8      InternetAddress[] toEmailAddresses =
9          InternetAddress.parse("user1@gmail.com, user2@gmail.com");
10     InternetAddress[] ccEmailAddresses =
11         InternetAddress.parse("user21@gmail.com, user22@gmail.com");
12     InternetAddress[] bccEmailAddresses =
13         InternetAddress.parse("user31@gmail.com");
14
15     message.setRecipients(Message.RecipientType.TO, toEmailAddresses);
16     message.setRecipients(Message.RecipientType.CC, ccEmailAddresses);
17     message.setRecipients(Message.RecipientType.BCC, bccEmailAddresses);

```

```
18
19  /* Mail body with plain Text */
20  message.setText("Hello User,"
21                  + "\n\n If you read this, means mail sent with Java Mail API is successful");
22
```

Para enviar un correo electrónico en texto sin formato, simplemente pasamos el texto en el método `message.setText()`.

## Enviar mensaje

Hasta ahora, hemos creado una sesión y redactado el mensaje. Ahora es el momento de enviar el mensaje a los destinatarios. Usaremos la clase `jakarta.mail.Transport` para hacerlo. La clase proporciona métodos `send()` sobrecargados.

```
1  // Send the configured message in the session
2  Transport.send(message);
```

java

### Emails temporales

**YopMail** (<http://www.yopmail.com/en/>) es un servicio de correo electrónico temporal que se puede utilizar para recibir correos electrónicos de forma anónima. No se requiere registro para usar el servicio. Puede usar cualquier dirección de correo electrónico de su elección para recibir correos electrónicos. El servicio es gratuito y se puede utilizar para recibir correos electrónicos de forma anónima. El servicio es gratuito y se puede utilizar para recibir correos electrónicos de forma anónima.

## 5.1.4 Enviar mensajes HTML

En términos de usabilidad, el contenido HTML es muy superior al texto sin formato. Por lo tanto, la mayoría de las veces enviamos emails en formato HTML. Java Mail API admite el envío de correos electrónicos en formato HTML. Para enviar un correo electrónico con contenido HTML, debemos reemplazar el método `message.setText()` con el siguiente código.

```
1  ...
2  message.setContent("Just discovered that Jakarta Mail is fun and easy to use",
3                      "text/html");
4  ...
```

java

usaremos el método `setContent()` para establecer el contenido y especificar `"text/html"` en el segundo argumento, lo que indica que el mensaje tiene formato HTML.

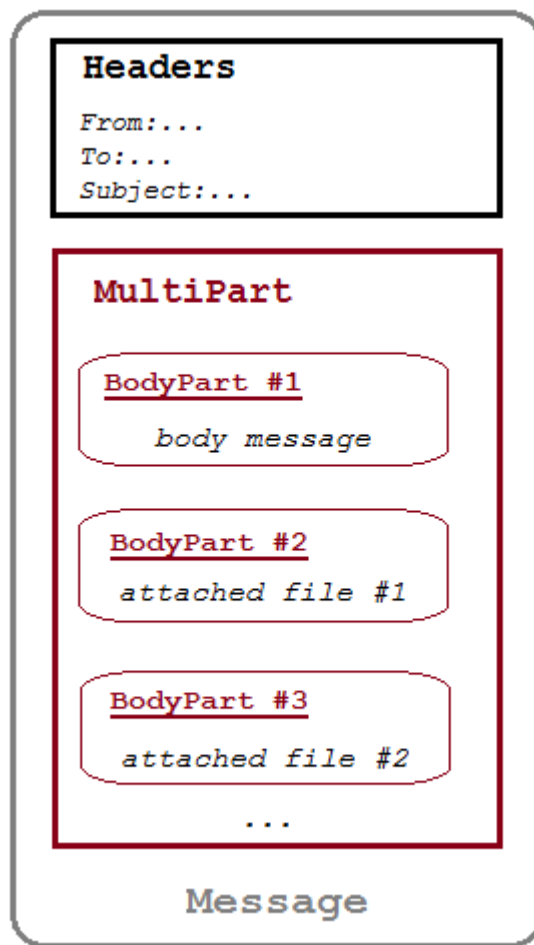
## 5.1.5 Enviar correo electrónico con archivos adjuntos

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:

1. 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.
2. Luego, establecemos la primera parte del objeto *Multipart* pasándole el mensaje real.



3. A continuación, establecemos la segunda y las siguientes parte del objeto *Multipart* agregando los datos adjuntos.
4. Incluimos el objeto *Multipart* en el mensaje a enviar.
5. Enviamos el mensaje



```

1  // create an instance of multipart object
2  Multipart multipart = new MimeMultipart();
3
4  // create the 1st message body part
5  MimeBodyPart messageBodyPart = new MimeBodyPart();
6  // Add a plain message (HTML can also be added with setContent)
7  messageBodyPart.setText("Please find the attachment sent using Jakarta Mail");
8  // Add the BodyPart to the Multipart object
9  multipart.addBodyPart(messageBodyPart);
10
11 // 2nd. bodyPart with an attached file
12 messageBodyPart = new MimeBodyPart();
13 String filename = "C:/temp/file1.pdf";
14 messageBodyPart.attachFile(filename);
15 // Add the BodyPart to the Multipart object
16 multipart.addBodyPart(messageBodyPart);
17
18 // Add the multipart object to the message
19 message.setContent(multipart);
20
21 // Send the message with multipart MIME objects
22 Transport.send(message);

```

java

## 5.1.6 Enviar correos electrónicos HTML con imágenes

Para agregar una imagen a su correo electrónico HTML en Jakarta Mail, puede elegir cualquiera de las tres opciones comunes:

- Insertando imágenes CID.
- Insertando imágenes en línea o codificación Base64.
- Enlazando imágenes.

### Insertando imágenes CID (Content-ID)

Para insertar imágenes CID, debemos crear un mensaje `MIME multipart/related` usando el siguiente código:

```
1 // 1st part of the message. An HTML code with a CID referenced image
2 Multipart multipart = new MimeMultipart("related");
3 MimeBodyPart htmlPart = new MimeBodyPart();
4 //add reference to your image to the HTML body 
5 String messageBody = "<p></p><img src=\"cid:my-test-image-cid\" alt=\"embedded img\" /></p>";
6 htmlPart.setText(messageBody, "utf-8", "html");
7 // Add the BodyPart to the Multipart object
8 multipart.addBodyPart(htmlPart);
9
10 // 2nd part of the message. The image with special CID header markers
11 MimeBodyPart imgPart = new MimeBodyPart();
12 // imageFile is the file containing the image
13 imgPart.attachFile(imageFile);
14 // or, if the image is in a byte array in memory, use
15 // imgPart.setDataHandler(new DataHandler(
16 //     new ByteArrayDataSource(bytes, "image/whatever")));
17 imgPart.setContentID("<my-test-image-cid>");
18 // Add the multipart object to the message
19 multipart.addBodyPart(imgPart);
20
21 // Add the multipart object to the message
22 message.setContent(multipart);
23
24 // Send the message with multipart MIME objects
25 Transport.send(message);
```

java

### Insertando imágenes en línea (Base64)

Para la inserción en línea o la codificación Base64, debe incluir los datos de imagen codificados en el cuerpo HTML de forma similar a esta:

```
1 
```

html



#### tamaño del HTML

Cada dígito Base64 representa 6 bits de datos, por lo que el código generado para una imagen real será bastante grande.

Como esto afecta el tamaño total del mensaje HTML, es mejor no usar imágenes grandes en línea.

Para codificar/decodificar una secuencia en Base 64 podemos usar la clase `java.util.Base64`

```
1 byte[] fileContent = new FileInputStream(imageFile).readAllBytes();
2 String base64EncodedData = Base64.getEncoder().encodeToString(fileContent);
```

java

### Codificación base64

Base64 es una buena opción para enviar datos binarios a través de protocolos de texto como HTTP, sin pérdida de información.

Esta operación se puede aplicar a cualquier archivo binario. Es útil cuando necesitamos transferir contenido binario en formato JSON, por ejemplo desde la aplicación móvil al endpoint de conexión REST.

[Image to Base64 String Conversion - Baeldung \(https://www.baeldung.com/java-base64-image-string\)](https://www.baeldung.com/java-base64-image-string)

## Linked images

Por último, tenemos imágenes vinculadas que son esencialmente imágenes alojadas en algún servidor externo al que luego crea un enlace. Podemos hacerlo usando la etiqueta `img` en el cuerpo HTML de la siguiente manera.

```
1 
```

html

### Debug Jakarta Mail

La depuración juega un papel fundamental en las pruebas de envío de correo electrónico.

En Jakarta Mail, es bastante sencillo. Tan solo hay que establecer debug a true en las propiedades de configuración de la sesión:

```
props.put("mail.debug", "true");
```

Como resultado, obtendremos una descripción paso a paso de cómo se ejecuta el código. Si aparece algún problema con el envío del mensaje, comprenderemos instantáneamente lo que ha sucedido y en qué paso.

## 5.1.7 Lectura de emails con Jakarta Mail

El API de correo de Jakarta también proporciona soporte para leer correos electrónicos. Para leer correos electrónicos, debe usar la clase `javax.mail.Store`. La clase `Store` es una clase abstracta que modela un almacén de mensajes y su protocolo de acceso, y es una subclase de las clases `POP3Store` e `IMAPStore`.

Leer correos electrónicos almacenados en un servidor IMAP consta de los siguientes pasos:

- Creación de la sesión IMAP (Session), indicando el protocolo, el nombre del host, el puerto, si usa SSL y el servidor de autenticación asociado.
- Configuración y obtención del almacén (Store).
- Obtención de la conexión a través del almacén, indicando el identificador de la cuenta y la contraseña.

- Obtención de la carpeta a leer.
- Apertura de la carpeta.
- Obtención de los mensajes
- Procesamiento de mensajes
- Cierre de la carpeta y del almacén.
- Cierre de la sesión y de la conexión

```
1 // Prepare IMAP configuration into a Property object
2 final Properties prop = new Properties();
3 prop.put("mail.imap.host", "imap.gmail.com");
4 prop.put("mail.imap.port", "993");
5 prop.put("mail.imap.ssl.enable", "true");
6 prop.put("mail.imap.auth", "true");
7
8 // Create the Session with the user credentials
9 Session mailSession = Session.getInstance(prop, new jakarta.mail.Authenticator() {
10     @Override
11     protected PasswordAuthentication getPasswordAuthentication() {
12         return new PasswordAuthentication(prop.getProperty("mail.imap.username"),
13             prop.getProperty("mail.imap.password"));
14     }
15 });
16
17 // Get the Store object and connect to the current host using the specified username and password.
18 Store store = mailSession.getStore("imap");
19 store.connect(prop.getProperty("mail.imap.host"),
20     prop.getProperty("mail.imap.username"),
21     prop.getProperty("mail.imap.password"));
22
23 // Get the folder and open it
24 Folder folder = store.getFolder("INBOX");
25 folder.open(Folder.READ_ONLY);
26
27 // Get the messages
28 Message[] messages = folder.getMessages();
29
30 // Process the messages
31 for (int i = 0; i < messages.length; i++) {
32     Message message = messages[i];
33     System.out.println("Message " + (i + 1));
34     System.out.println("From: " + message.getFrom()[0]);
35     System.out.println("Subject: " + message.getSubject());
36     System.out.println("Sent Date: " + message.getSentDate());
37     System.out.println("Text: " + message.getContent().toString());
38 }
39
40 // Close the folder and store objects
41 folder.close(false);
42 store.close();
```

java