ALSIE CONSULTORES PEDAGOGICOS S.R.L.

PATRONES DE DISEÑO

BÚSQUEDA EN MENSAJES DE CORREO ELECTRÓNICO USANDO LOS PATRONES ITERATOR Y VISITOR

Elaborado por

Luis Leonardo Mejillones Rodriguez

Cochabamba – Bolivia

Descripción

La aplicación que se desarrolló en un cliente de correo electrónico cuya finalidad principal es la de enviar y recibir mensajes de correo electrónico. Para fines prácticos, el producto obtenido es una versión reducida cuya finalidad es la de realizar búsquedas entre los correos electrónicos almacenado localmente.

Por otra parte, tampoco se adiciona la funcionalidad de envío y recepción hacia servidores de correo electrónico reales, simplemente se llenan los folders de la aplicación con correos falsos.

El la figura 1 puede verse el diagrama de clases que intervienen en la solución planteada.

La interfaz **Folder** y su implementación, **FolderImpl**, se encargan de representar a cada folder en la aplicación y dentro ésta se almacenan instancias de la interfaz **Message** y su implementación **MessageEmail** cuya finalidad es almacenar los correos electrónicos.

Como puede verse en la figura 1, ambas interfaces, **Folder** y **Message** heredan de una interfaz común denominada Element, esto permite disponer de un mecanismo común para poder englobar a todos los elementos que intervienen en la aplicación. La técnica de definición de las interfaces mencionadas siguen los principio SOLID[1].

En la parte de búsquedas, la interface **Folder** se compone de las interfaces **EmailIterator** y Visitor, los cuales implementan los patrones Iterator y Visitor respectivamente. Ambas implementaciones seŕan descritas más adelante.

Para el desarrollo de este ejercicio se utilizó el lenguaje de programación Java y se aplicó el estándar de codificación definido por google denominado Google Java Style.

Patron Iterator

El patrón Iterator tiene la finalidad de permitir el recorrido a través de listas de elementos sin exponer su estructura interna. En la aplicación se hace necesario recorrer la lista de correos presentes en cada folder, es decir que el patrón se encarga de recorrer primeramente la lista de folders de la aplicación y por cada folder recorrer la lista de mensajes que contiene. En la figura 2 puede verse con más detalle, solo la parte de implementación del patrón Iterador.

La interfaz **EmailIterator** define tres métodos, uno para saber si existen más elementos (**hasNext**) otra para obtener el elemento siguiente (**next**) y finalmente un método para obtener el elemento actual sin mover el índice. En la clase que implementa esta interfaz, la clase **EmailIteratorImpl** define la lista de elementos y un índice que se encarga de almacenar la ubicación del elemento vigente.

[1] Acrónimo que define los principios de diseño y desarrollo orientado a objetos.

Cada vez que se llama el método **getIterator** de la interfaz Folder, se devuelve una instancia del iterador.

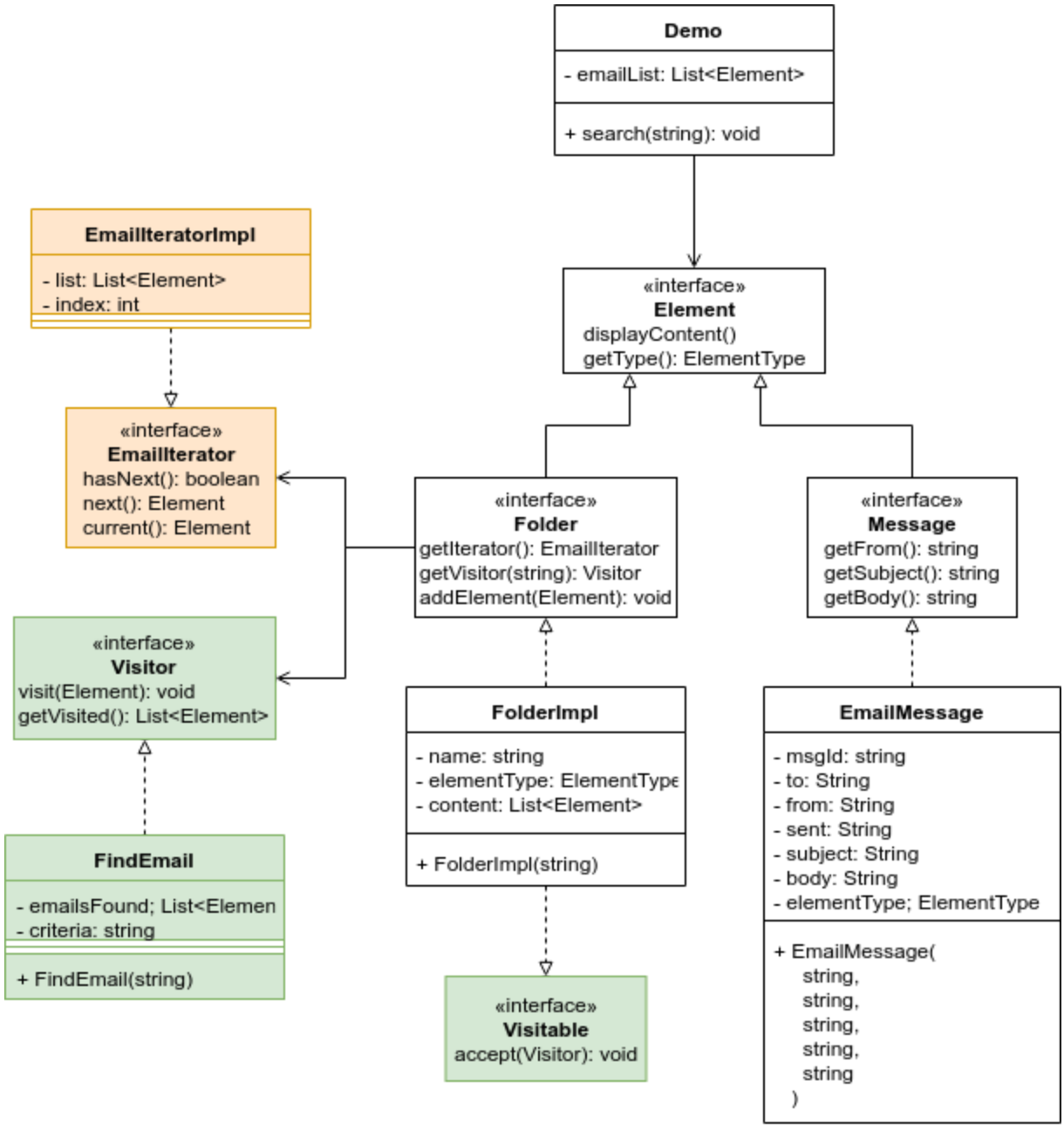

Figura 1. Diagrama de clases de la aplicación

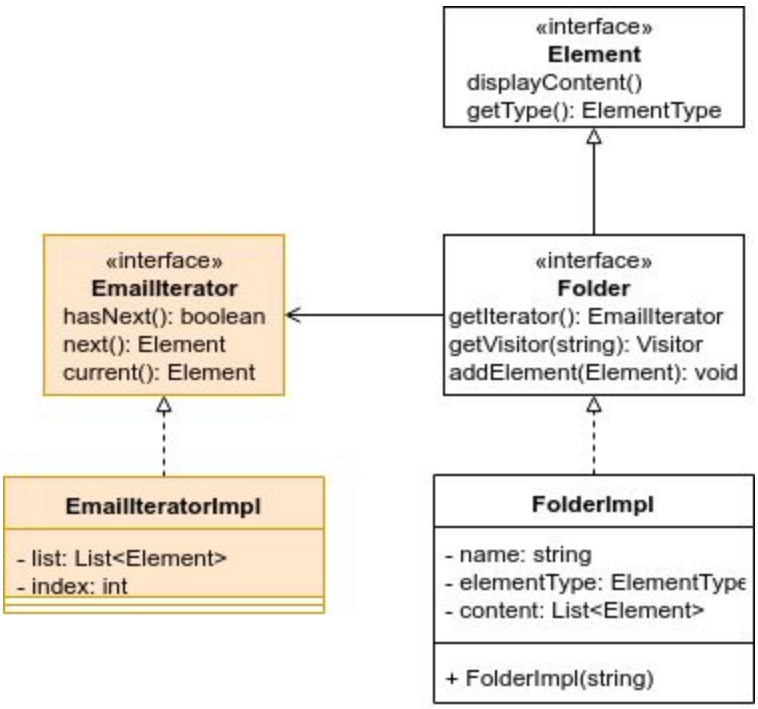

Figura 2. Diagrama de clases de la implementación del patrón Iterator

Patron Visitor

El patrón Visitor es encargado de realizar alguna tarea con cada elemento susceptible de ser visitado en algún tipo de lista de elementos. En la aplicación se utilizó el patrón **Visitor** para saber si cada mensaje de correo contiene el texto buscado, de ser así, es almacenado en otra lista de elementos encontrados.

En la figura 3 puede verse el detalle de la implementación del patrón **Visitor** en la aplicación.

El patrón visitor necesita de dos interfaces, una encargada de permitir la visita a cada elemento y la otra proveer la facultad de ser visitado a las objetos que podrán ser visitados, como se ve en la figura 3, la clase **FolderImpl** implementa también la interfaz **Visitable**, cuyo único fin es permitir aceptar a cada elemento para ser visitado.

Finalmente, en la interfaz **Visitor**, también se definió un método llamado **getVisited** que simplemente retorna el resultado de los elementos visitados, en este caso la lista de correos que cumplen el criterio de busqueda.

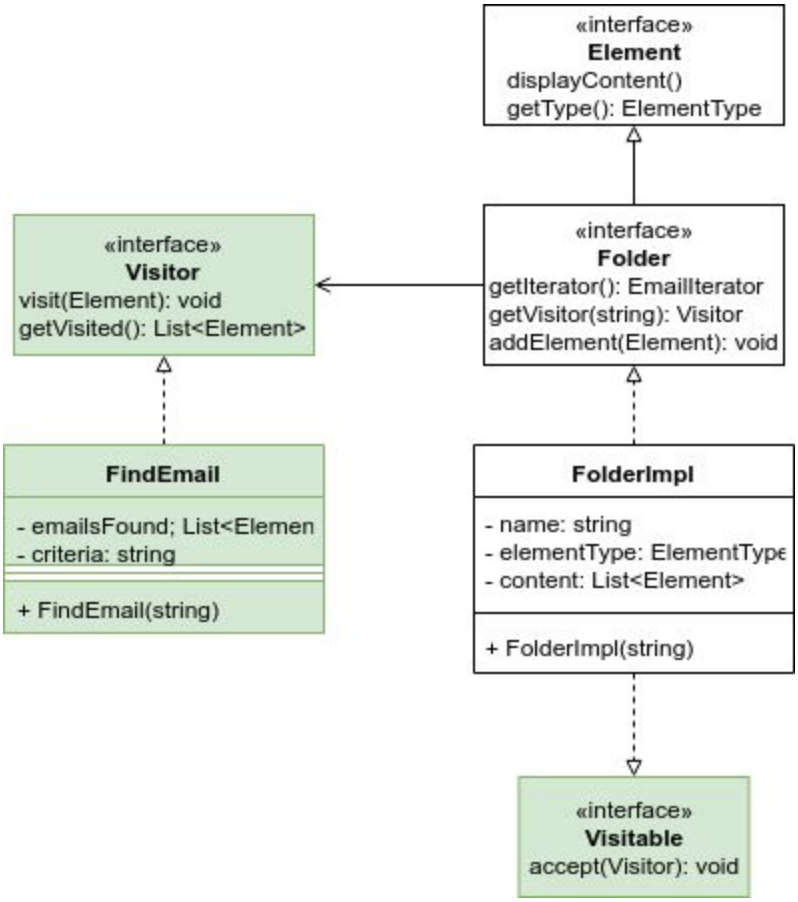

Figura 2. Diagrama de clases de la implementación del patrón Iterator

Conclusiones

El uso de patrones de diseño facilita enormemente la planificación y desarrollo de soluciones de software, permitiendo conseguir productos altamente flexibles y mantenibles en el tiempo. En el ejercicio desarrollado se vio el poder que tienen éstos al ser combinados para obtener resultados altamente positivos. Sin duda, la utilización de los patrones, facilita grandemente e la vida de un desarrollador cuando son empleados eficientemente.

Referencias

https://google.github.io/styleguide/javaguide.html
https://dzone.com/articles/design-patterns-visitor
https://informaticapc.com/patrones-de-diseno/iterator.php
https://en.wikipedia.org/wiki/SOLID