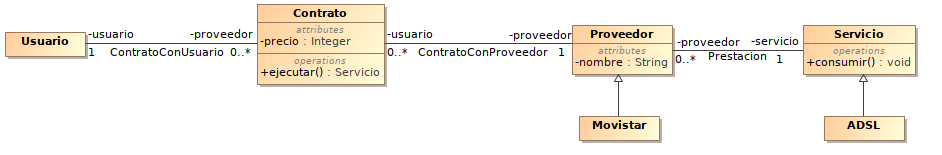
**Práctica 4**

**Ejercicio 1**

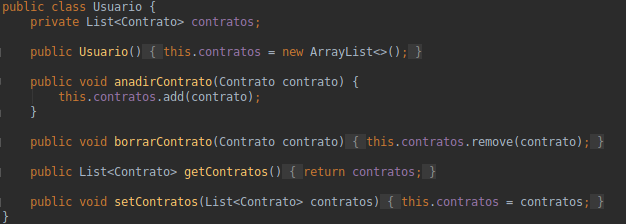
Puesto que la clase Proveedor es abstracta, desde la clase Contrato no se puede acceder a la clase Servicio. Cada Proveedor tiene asociado un solo Servicio, y al ser bidireccional la relación, tenemos un atributo de tipo Servicio en la clase Proveedor, pudiendo llegar a este atributo desde Contrato.

Para ello, habría que transformar la clase de asociación Contrato en una clase entre Usuario y Proveedor manteniendo la consistencia de la relación desde Contrato.

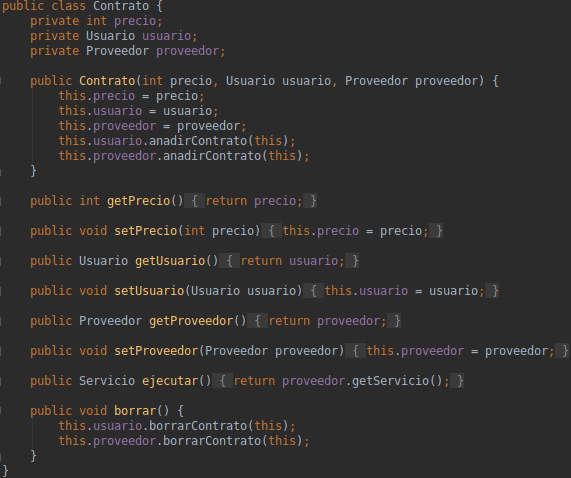


Siguiendo este diagrama, la implementación sería la siguiente:

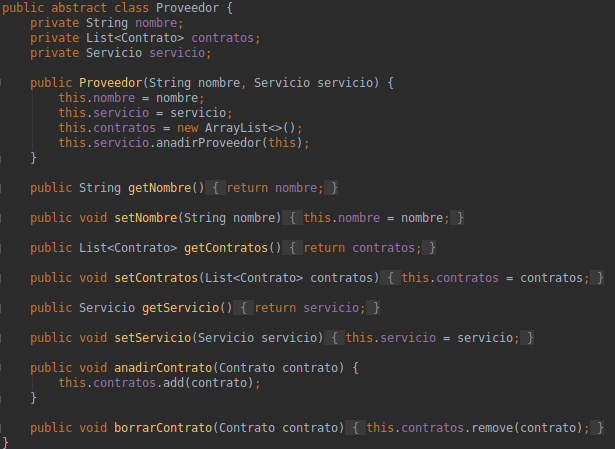
* **Usuario.**

****

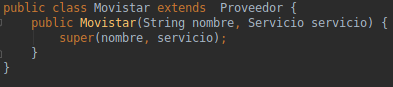
* **Contrato.**

****

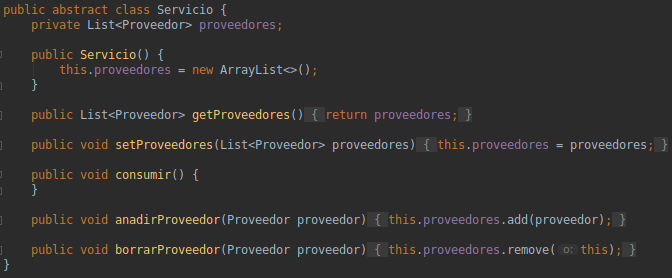
* ***Proveedor*.**

****

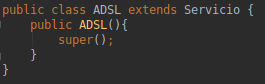
* **Movistar**, que hereda de *Proveedor*.



* ***Servicio***.

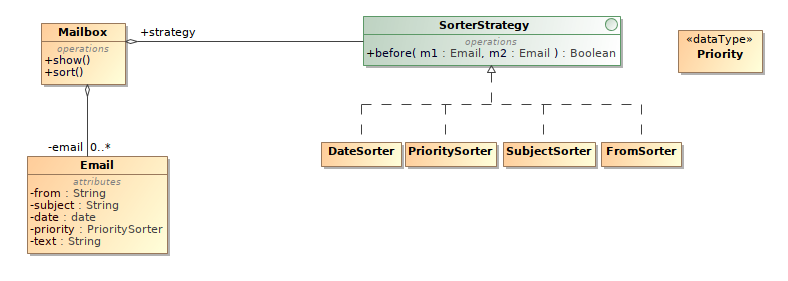


* **ADSL**, que hereda de *Servicio*.



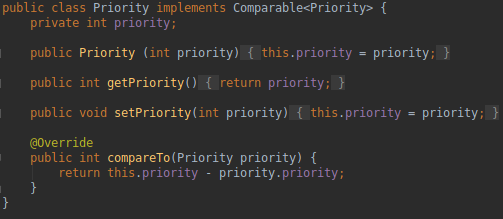
**Ejercicio 2**

Para poder ordenar los correos mostrados en la clase Mailbox mediante distintos métodos pero sólo uno a la vez utilizamos el patrón de diseño Strategy, implementando una estrategia distinta del método *before* para cada método de ordenación (suponiendo que no queremos ordenar los emails por su contenido o texto).



Siguiendo el diagrama, la implementación sería la siguiente:

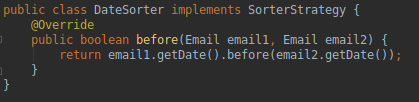
* **Priority**

****

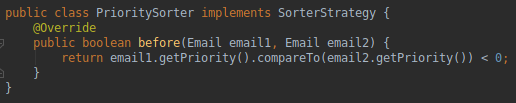
* ***SorterStrategy***

****

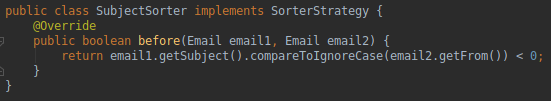
* **DateSorter**

****

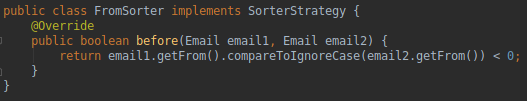
* **PrioritySorter**

****

* **SubjectSorter**

****

* **FromSorter**



* **Mailbox**

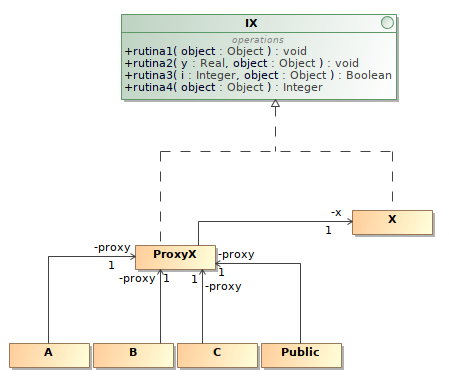
****

* **Email**



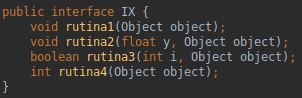
**Ejercicio 3**

1. La exportación selectiva no es posible realizarla en Java, aunque se puede usar el patrón Representante (Proxy). Este patrón define una clase intermedia para que se encargue de administrar las clases que pueden hacer uso de determinadas funciones de otra clase compartiendo la interfaz para así definir las operaciones. Sin embargo, Eiffel si permite administrar qué funciones son utilizadas por cada clase.
2. Como ya hemos comentado, se puede utilizar el patrón Proxy para realizar una implementación de la exportación selectiva en Java. La solución tiene las mismas ventajas que la exportación selectiva de Eiffel, pero tiene algunas desventajas, por cada clase u operación a realizar, se debe especificar dentro de la clase Proxy y la seguridad para verificar qué clase hace qué función es bastante baja. Así pues, la solución sería la siguiente.

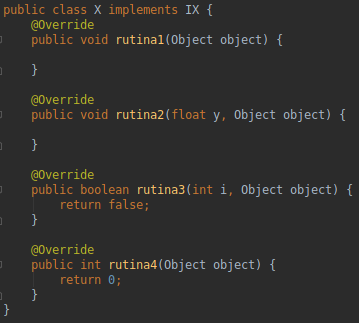


Siguiendo este diagrama, la implementación sería la siguiente (los métodos de las rutinas quedarían vacíos puesto que en el enunciado no se indica la funcionalidad):

* **Interfaz *IX***

****

* **X** implementa *IX*



* **A**

****

* **B**

****

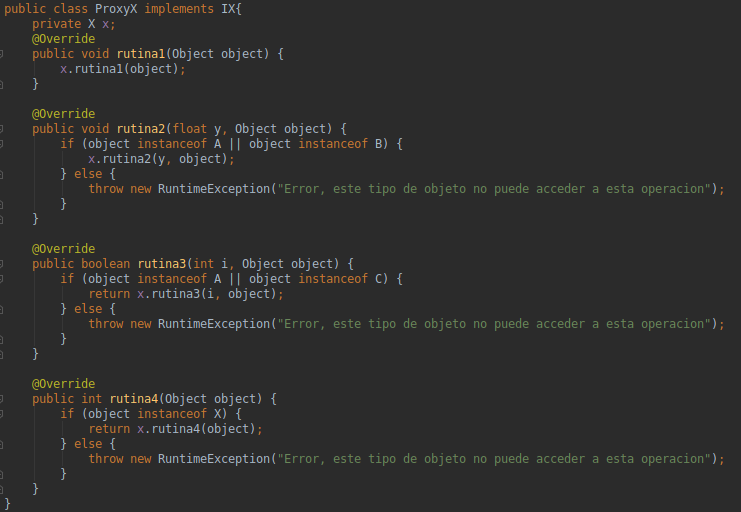
* **C**

****

* **Public.**

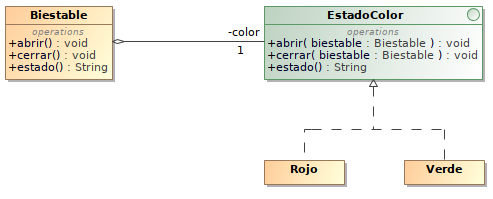


* **ProxyX**, implementa *IX*



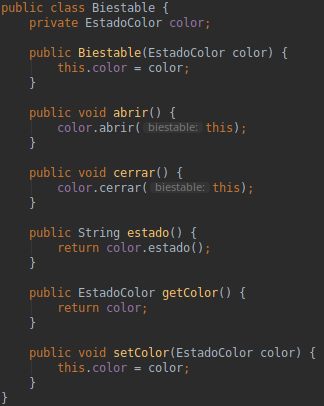
**Ejercicio 4**

1. Para este ejercicio utilizamos el patrón de Estado ya que tenemos un Biestable con dos estados. Para la implementación, hacemos que el Biestable tenga una variable del tipo *EstadoColor* que puede ser tanto del tipo Rojo como Verde (estados del Biestable).

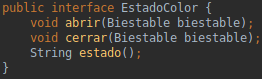


Siguiendo este diagrama, la implementación sería la siguiente:

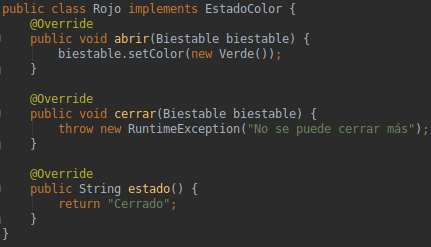
* **Biestable**



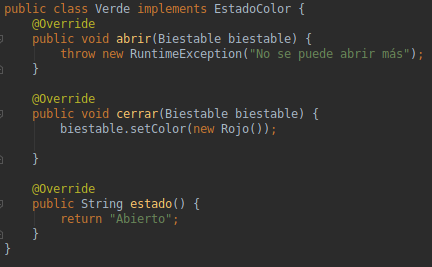
* ***EstadoColor***



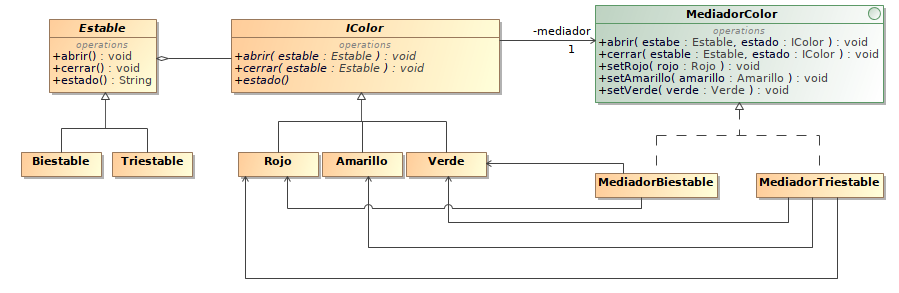
* **Rojo**



* **Verde**



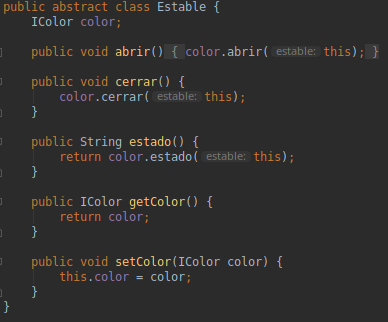
1. Reutilizando la respuesta al apartado anterior y añadiendo comprobación de tipo y las Clases *Estable*, Biestable, Triestable y Amarillo.



La clase abstracta *Estable* contiene la definición de los métodos abrir, cerrar y estado al igual que la clase Biestable antes. Biestable y Triestable heredan de la clase Estable. Se ha sustituido la interfaz *IColor* por la clase abstracta para poder introducir el atributo *IMediadorColor* que afecta a los estados.

Siguiendo este diagrama, la implementación sería la siguiente:

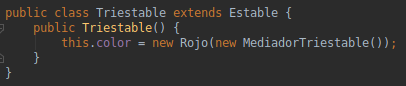
* ***Estable***

******

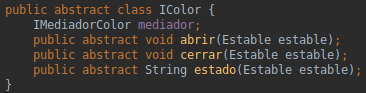
* **Biestable**

****

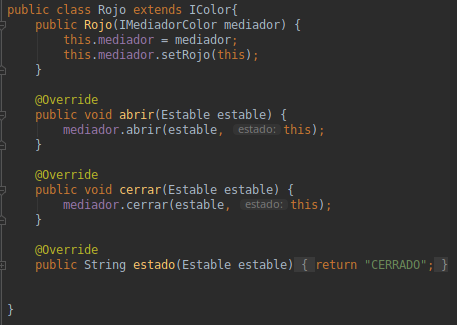
* **Triestable**

****

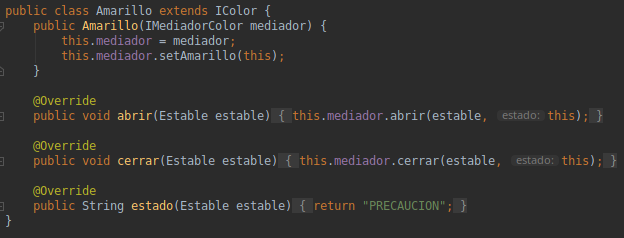
* ***IColor***

****

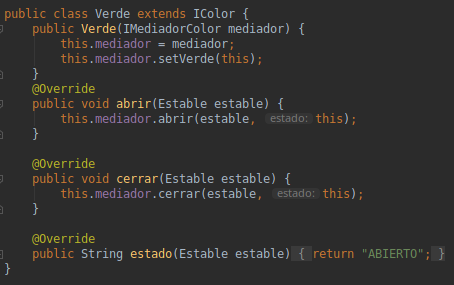
* **Rojo**

****

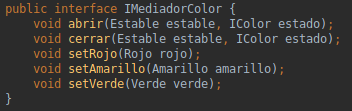
* **Amarillo**

****

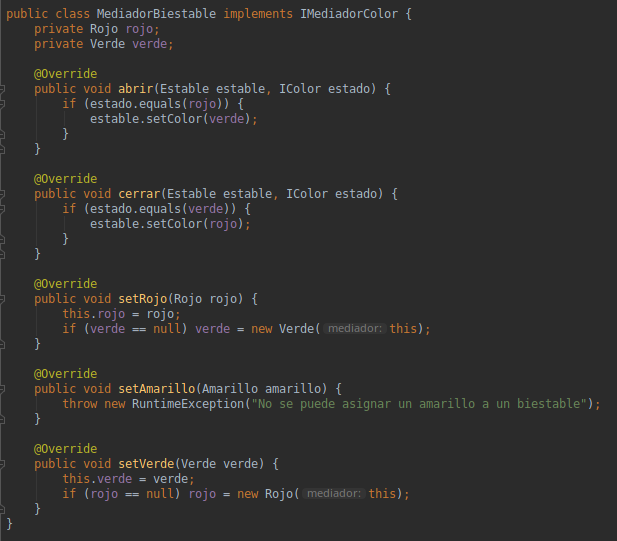
* **Verde**

****

* **IMediadorColor**

****

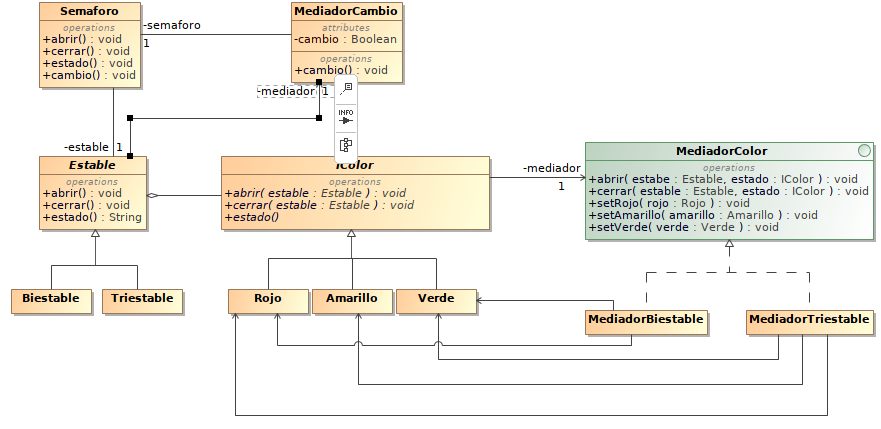
* **MediadorBiestable**

****

* **MediadorTriestable**

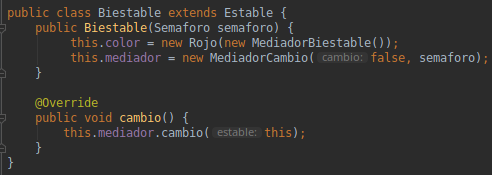


1. Introducimos la clase Semaforo que contiene un atributo de tipo Estable para realizar el cambio del tipo del atributo. Para esto necesitamos un método setMediador en la clase IColor que recibe como parámetro un mediador para poder mantener el estado en el que nos encontramos.

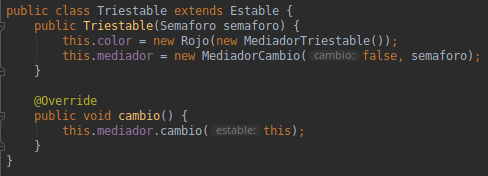


Siguiendo este diagrama, la implementación sería la siguiente (las clases que no aparecen no han sufrido cambios respecto el apartado anterior):

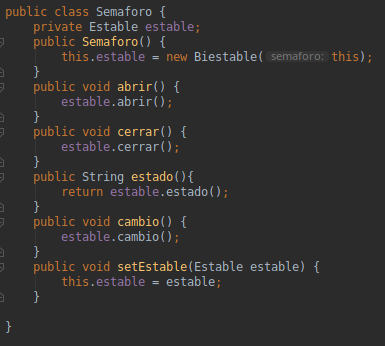
* **Biestable**

****

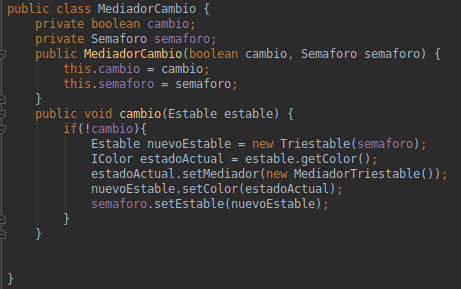
* **Triestable**

****

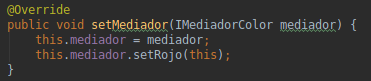
* **Semaforo**

****

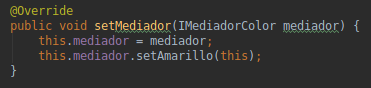
* **MediadorCambio**

****

* **Rojo**

****

* **Amarillo**

****

* **Verde**

