Práctica 2

UGR

Desarrollo de Software

Autores:

- Alba Ayala Carmona
- María Ramos Martínez
- Sofía González Uceda
- Natalia Serrano Cerceda

Índice

1. Ejercicio 1

1.1. Explicación

En el ejercio 1 se nos pedía realizar un mantenimiento del ejercicio 4 de la prática anterior. Los cambios son los siguientes:

1.1.1. Mantenimiento adaptativo

Para cambiar de lenguaje de programación, de Java a Dart/Flutter, hemos tenido que adaptar todas las clases al nuevo lenguaje, sobre todo el main.

Uno de los cambios que hemos hecho es que los filtros, en vez de devolver un bool, devuelven una pareja formada por un *string* que contiene un mensaje de error o éxito, y un booleano (que ya devolvía antes), para saber si se ha pasado o no el filtro. Con este cambio también hemos tenido que adaptar la clase *filterChain*, *filterManager* y *Cliente*, para que devuelvan un string con el mensaje correspondiente.

En el main seguimos creando los objetos necesarios y, además, hemos añadido un método privado *_createCuenta* que se invoca cuando se pulsa el botón en el que se crea al cliente y luego se crea la cuenta con el correo y contraseña que introduzca el usuario en los *TextField* corespondientes. Con el botón se comprueba que los campos de texto tengan contenido, en caso positivo se intenta crear la cuenta. Para ello, hemos tenido que comprobar que ninguno esté vacío antes de intentar crearla. En el caso de que todo vaya bien, al pulsar el botón se mostrará una lista con los filtros que ha pasado y cuáles no para, finalmente, indicar si la cuenta ha sido creada o no.

1.1.2. Mantenimiento perfectivo

Para el filtro que comprueba el número de carácteres de la contraseña, en la clase *Cuenta*, hemos añadido un método que nos devuelve directamente la longitud de la cadena, en vez de tener que obtener la contraseña y luego comprobar su longitud. También hemos puesto el _tamMin como una variable de la clase, en vez de crearla en el método.

Para el filtro de la mayúscula, hemos cambiado el *if* que hacía dos comprobaciones por una expresión regular que compruebe si es una letra mayúscula. Para el filtro que verifica que al menos haya un número, hemos cambiado también la comprobación del *if* por otra expresión regular, para simplificarlo.

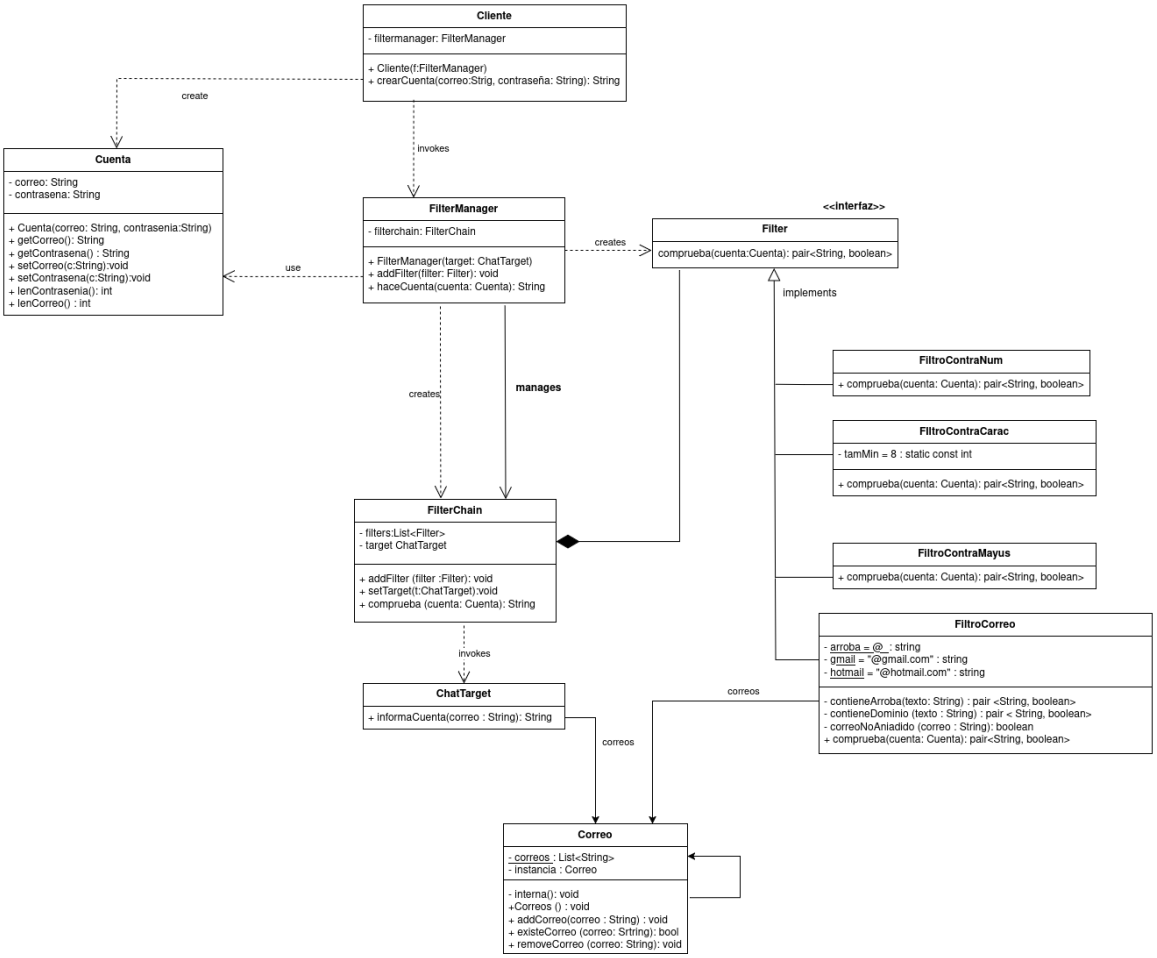
Para el filtro del correo hemos corregido los errores que se nos indicó en la prática anterior (crear variables constantes detro de un *for*, lo hemos modificado a atributos de clase). Además, hemos separado las distintas comprobaciones que se hacen en el *for* en dos métodos distintos, para mayor legibilidad. Así mismo, para comprobar que esté la '@', ya no nos hace falta un *for* que recorra cada carácter, ya que con el método *contains* de la clase *string* comprobamos si está. Para ver si contiene el dominio después de '@' y que este sea corecto hacemos lo mismo, usamos el método *endsWith*, para ver si la terminación del correo es válida, lo que hace que el método dominio de la prática 1 quede en desuso.

1.1.3. Nueva funcionalidad: comprobar si el correo existe

Hemos añadido la clase *Correos*, que sigue un patrón *Sigleton*, para acceder a la misma lista de correos desde las clases necesarias. Los cambios que esto ha supuesto en el filtro correo son: tener una instacia de Correos y preguntarle si el correo introducido existe ya o no.

Para añadir un nuevo correo a la lista que contiene Correos, se hace en la clase *ChatTarget*, a la que solo se invoca si se pasan todos los filtros de forma correcta.

1.2. UML



(a) Nuevo patrón

2. Ejercicio 2

2.1. Explicación

Este ejercicio consiste en crear una aplicación con Dart/Flutter para conectarse a la API de Hugging Face. Mediante esta aplicación el usuario puede seleccionar un LLM desde la interfaz y, a partir del mensaje que introduzca, obtener la respuesta correspondiente.

2.1.1. Conexión con Hugging Face

Para conectarnos con Hugging Face desde Dart/Flutter hemos consultado el siguiente enlace de la documentación oficial de Flutter: https://docs.flutter.dev/cookbook/networking/authenticated-requests.

Para conectarnos a la API es necesario un token, que se carga desde un archivo (*huggingface.txt*) y se incluye en los headers de la función *post* del paquete *http*, junto con la URL del modelo que se quiera consultar.

Dado que se trata de una operación asíncrona, el método se declara con la palabra clave `async`, y se utiliza `await` para esperar la respuesta de la función `post` del paquete `http`. El tipo de retorno `Future<String>` indica que la función devolverá un valor en el futuro, una vez que se haya completado la operación.

2.1.2. Patrón Strategy

Para la implementación de este ejercicio se ha usado el patrón *Strategy*, por lo que contamos con una clase *MyContext* que hace las llamadas a la estrategia elegida por el usuario. Las estrategias disponibles (*Basico*, *Traductor* y *Expandir*, asociadas a los modelos correspondientes) heredan de la clase *Strategy*. En esta clase estrategia hemos implementado el método que se encarga de hacer las peticiones al modelo correspondiente; como todos los modelos siguen el mismo procedimiento, solo cambia la URL propia del modelo. Esto es extensible ya que si se quisiera añadir otra estrategia distinta bastaría con sobreescribir el método *Future<String>AlgorithmInterface(String text) async{}*.

2.1.3. Uso de la aplicación

La aplicación utiliza dos archivos externos que se especifican en el archivo *pubspec.yaml* en la sección de *assests*:

```
assets:
    - huggingface.txt
    - modelos.json
```

En el archivo *huggingface.txt* se incluirá el token necesario para realizar las peticiones a los modelos.
En el archivo *modelos.json* se especifican los modelos a utilizar. Los modelos elegidos son los mismos que utilizamos en la práctica 1, que se describían como *Básico*, *Traductor* y de *Expansión*, por lo que en la interfaz de la aplicación hemos usado esa nomenclatura.

Estos dos archivos deben estar en la carpeta del ejercicio para el correcto funcionamiento de la aplicación (por ejemplo: *Ej2/huggingface.txt* y *Ej2/modelos.txt*).

2.2. UML

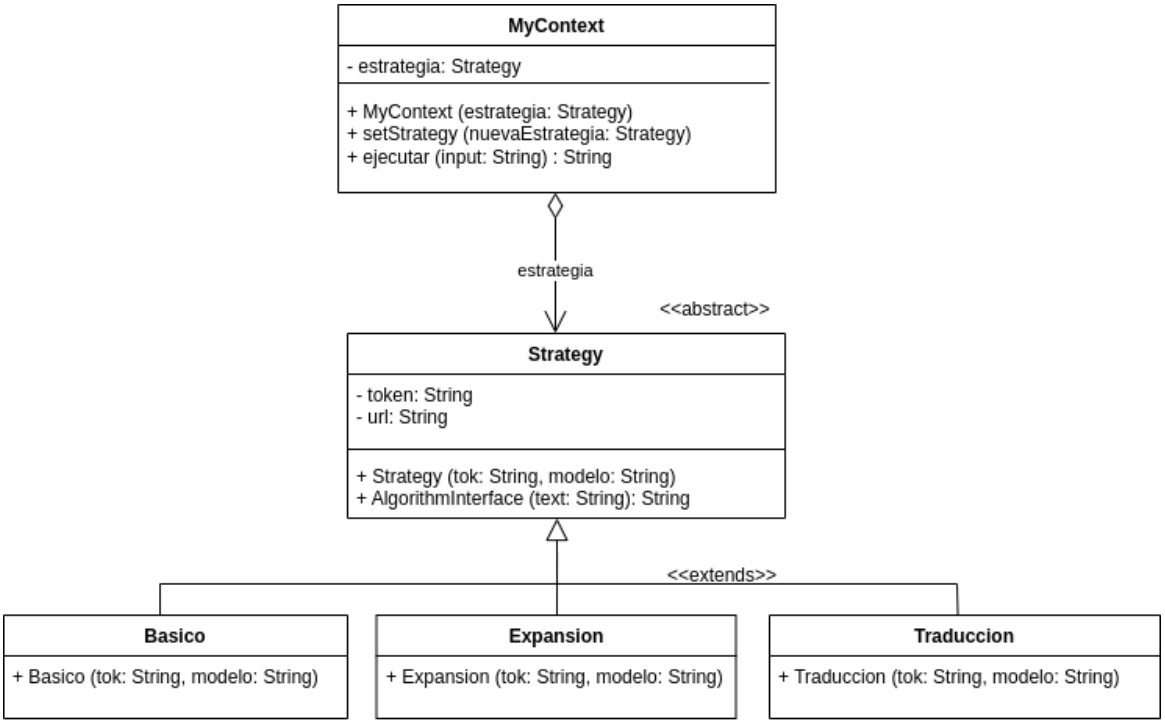


Figura 2: UML ejercicio 2