Trabajo Práctico Final
Programación con Objetos II
2do cuatrimestre 2022

Integrantes:

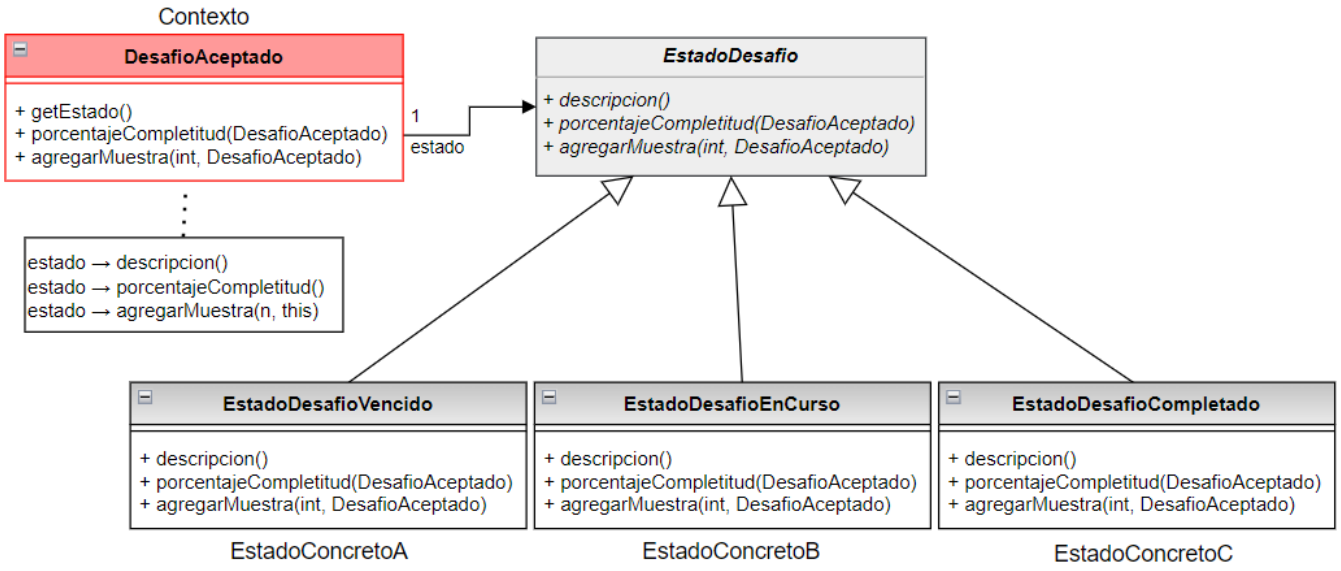
- Sergio Ruffinelli
- Ian Ghioni (ghioni2002@gmail.com)
- Chiara Forti Dono (chiarafdono@gmail.com)

Patrón State para los desafíos

La clase DesafioAceptado es una subclase de la clase Desafio. Esta última funciona como el desafío en sí mismo y una vez que ese desafío es aceptado por el usuario, se instancia la clase DesafioAceptado, cuya responsabilidad es contabilizar el desafío para ese usuario determinado.

El DesafioAceptado tiene ciertas características determinadas para el usuario, como la calificación, el momento en que se completa, la cantidad de muestras tomadas al momento, entre otras. Es decir, para una única instancia de Desafio, existen tantas instancias de DesafioAceptado que heredan de ella como usuarios hayan aceptado dicho desafio.

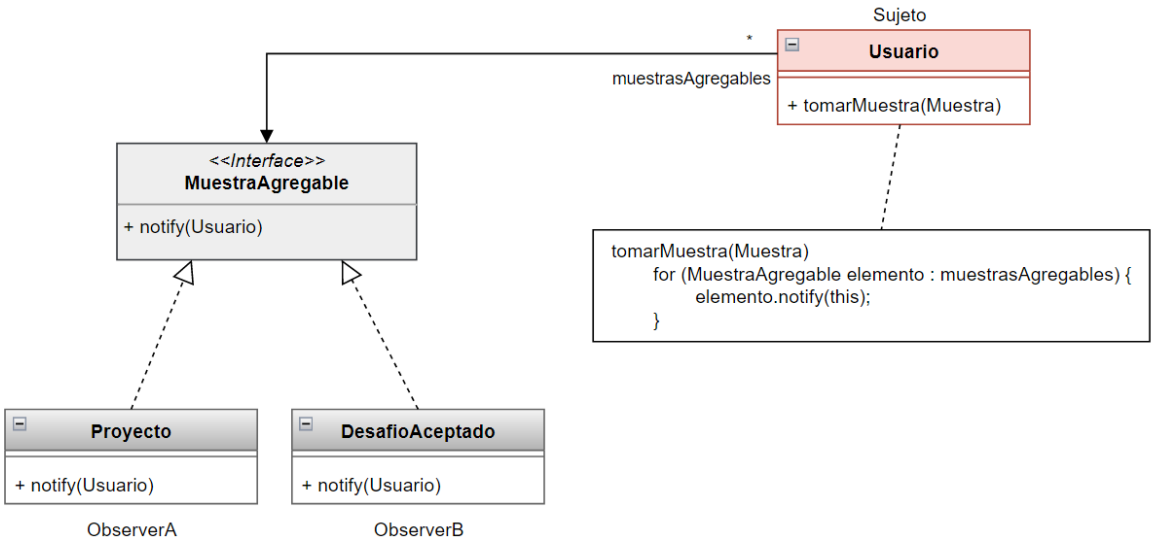
El comportamiento del DesafioAceptado depende del estado del mismo, ya que se necesita saber cuál es su estado para agregar muestras o calcular el porcentaje de completitud. Para esto, se implementó el patrón State, donde el contexto es el DesafioAceptado, el estado está tipado/representado con la clase abstracta EstadoDesafio y las clases concretas de EstadoDesafio son EstadoDesafioVencido, EstadoDesafioEnCurso y EstadoDesafioCompletado. De esta forma, el estado del desafío puede cambiarse de forma dinámica y transparente a medida que el usuario progresa con la realización del mismo.



Patrón Observer para los proyectos y desafíos

La interfaz MuestraAgregable es una interfaz que implementan las clases Proyecto y DesafioAceptado, ya que ambas tienen algo en común: cuando un usuario toma una muestra, deben actualizarse. Para el caso del proyecto, debe añadir la última muestra del usuario a su lista de muestras, y para el desafío, actualizar las muestras que se toman.

Se implementó entonces el patrón de diseño Observer, donde Usuario es el sujeto, la interfaz Observer es la interfaz MuestraAgregable, y los observadores concretos son Proyecto y Desafío. Cada vez que un usuario toma una muestra, se recorre una lista de objetos que implementan la interfaz MuestraAgregable, para notificarles que el usuario tomó una muestra, y cada objeto hará lo que tenga que hacer.

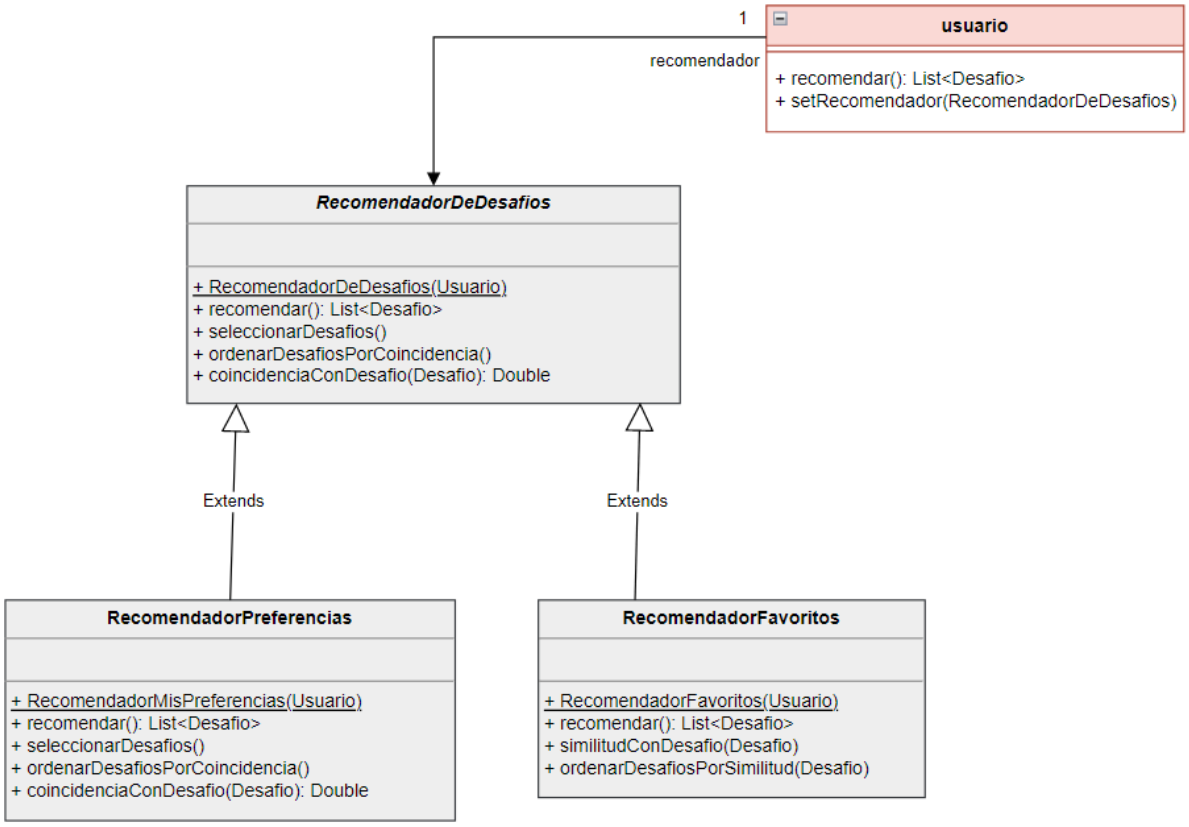


Si en un futuro se necesitara agregar otra clase que cuando el usuario tome una muestra, esta debe saberlo y actualizarse (Por ejemplo, otro elemento de ludificacion distinto a los Desafíos), es cuestión de simplemente crearla, que implemente la interfaz MuestraAgregable, y que el usuario lo añada a su lista de observers.

Patrón Strategy para el Recomendador De Desafios

Los recomendadores de desafíos concretos, extienden de la superClase abstracta RecomendadorDeDesafios. Esta tiene el método recomendar(), el cual las subclases concretas del recomendador se encargan de utilizar. En el caso del recomendador por preferencias, lo usa como ya esta implementado en la superclase y en el caso del recomendador por favoritos, lo sobreescribe para poder filtrar los desafíos en base a la estrategia de ese recomendador, pero reutilizando los metodos de la superclase que necesite reutilizar.
Los recomendadores de desafíos concretos son:

- RecomendadorPreferencias.
- RecomendadorFavoritos.

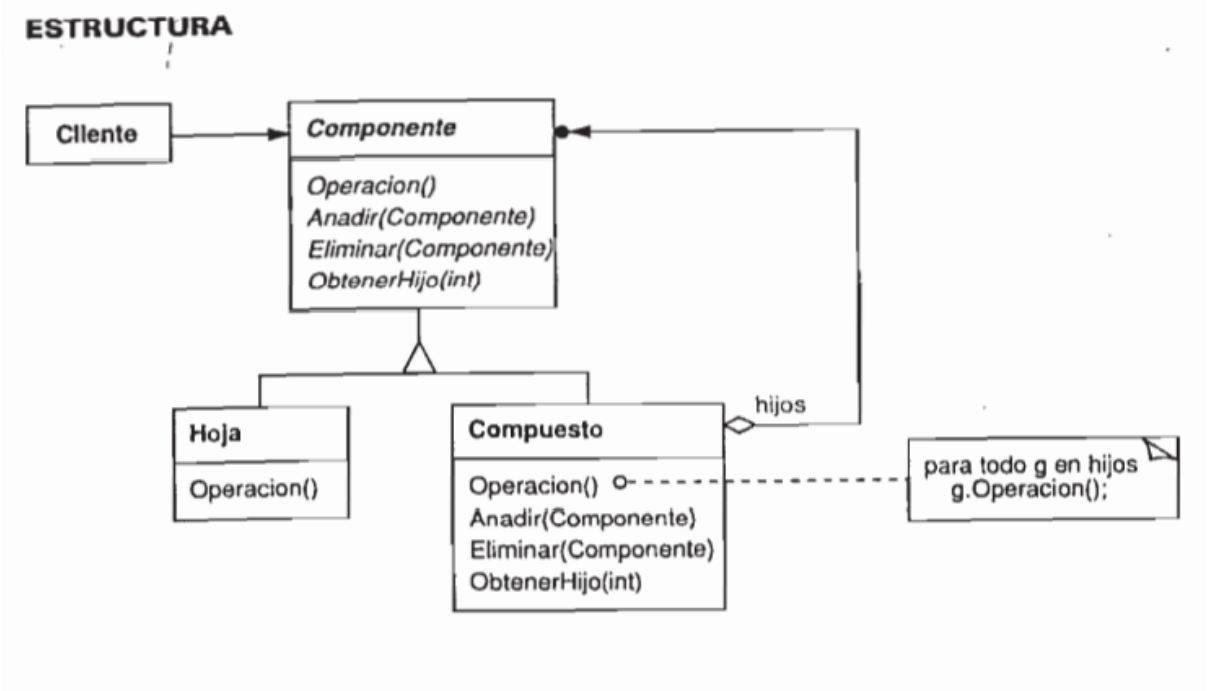


Un usuario puede entonces usar el recomendador de desafíos que tenga elegido en ese momento, y si quisiera cambiarlo por otro recomendador, puede hacerlo con el mensaje setRecomendador, que cambiara su recomendador de desafíos actual por otro nuevo.
En caso que se necesite una estrategia de recomendación distinta, solo basta con crearla y que extienda de RecomendadorDeDesafios, así el usuario puede utilizarla.

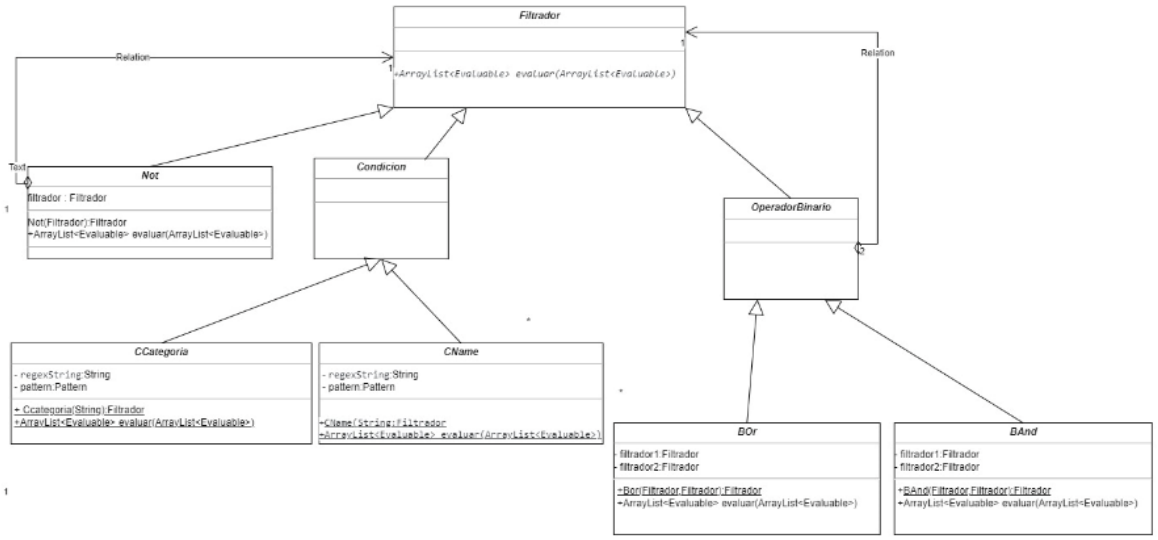
Patrón Composite para el Filtrador de Proyectos.

El filtrador de proyectos debe permitir componer diversas condiciones sobre las descripciones de los proyectos y las clases a las que pertenecen.
Para resolverlo implementamos el patrón Composite.
En esta gracias a la composición recursiva el filtrador puede tener como colaboradores uno o dos filtradores. Un colaborador en el caso del Not y dos en los casos de And y Or.
Estos colaboradores pueden ser tanto de las clases ya mencionadas como condiciones sobre los nombres o categorías de los proyectos.

Un diagrama generalizado para representar este patron puede ser el siguiente (Fuente Patrones de diseño - Erich Gamma et all)



En nuestra implementación del Filtrador la función de Hoja la cumplen las condiciones sobre el nombre o categoría de los proyectos.
La función de compuesto la cumplen el operador unario Not y los operadores binarios And y OR.



La operación que se ejecuta en todos los elementos de la estructura de árbol que compone un Filtrador es Evaluar.
Esta operación recibe una lista de Proyectos. Cada elemento de la estructura opera sobre la lista que recibe y retorna una nueva lista con los Proyectos que cumplen la condicion que representa el objeto.