
# Trabajo Integrador de Programación con Objetos 2

Integrantes: **Agustín Di Santo** **agusdisanto99@gmail.com**
**Santiago Abregú santiago_abregu_@hotmail.com,**
**Fernando Tomás Centurión ferc721@gmail.com**

Profesores: Diego Cano, Diego Torres y Matias Butti.

Fecha de entrega: 6/11/2022

# Dominio

El dominio habla sobre cada clase, cómo se implementa y cómo se relaciona con las demás, de ahí surgió la idea de implementar una aplicación móvil para el usuario.
Detalles importantes se hablan en el segundo párrafo, se mencionan las posibles nuevas actividades, aunque más a posterior se aclara que en éste trabajo hay una sola actividad, el desafío.
En la sección Proyectos se menciona el comportamiento y composición de un proyecto, pero algo a destacar es que con el equipo pensamos que la palabra “suscribir” hacía referencia al patrón Observer pero después nos dimos cuenta que no haría falta en este caso usar dicho patrón bajo el esquema que hemos planteado.

# Deuda técnica

- Tenemos una deuda técnica en la restricción temporal debido a que en el enunciado se podría implementar un patrón de diseño composite para poder filtrar mediante varias condiciones que tenía la restricción, una de ellas era que esté entre semana o fin de semana o inclusive mixtas tanto semana como fin de semana.

- Con respecto a las Actividades lúdicas, decimos que es una posible deuda técnica, en este caso no sería un error pero sí se tendría que mejorar en un futuro si queremos implementar más actividades, en sí en este trabajo no nos mencionan otras acciones que no sean desafíos pero podría implementarse para posibles casos de nuevas actividades a modo de acercarnos al principio Open-closed. Se implementó como una clase abstracta debido a que tiene el mensaje que todas entienden si no es un desafío, lo cual está bueno para poder mejorar el polimorfismo y poder filtrar por las actividades deseadas.

# Decisiones de modelo:

Se decidió delegar varias responsabilidades al usuario ¿Porqué decidimos implementar esta solución? Debido a que pensando en los principios Solid, ¿Había otras soluciones?
Claro, una de ellas era que el usuario contenga todas las implementaciones que tendría la aplicación.

Decidimos que el usuario tenga la responsabilidad de tener su propio estado, éste es el progreso del usuario que se vincula al desafío, cada usuario tiene un progreso independiente debido a que cada participante, progresa en el desafío de manera independiente, entonces dichos progresos que cada uno debe tener son individuales.

Decidimos tener diferentes recomendaciones para un usuario debido a que el usuario decide el algoritmo de recomendación que más desea y el sistema lo vincula con un conjunto de desafíos acorde a su preferencia decidida. Por eso implementamos el patrón de diseño Strategy debido a

que necesitamos un mediador en este caso el sistema, para que pueda encontrar los mejores desafíos en base a las recomendaciones del usuario.

Hay dos tipos de recomendación:
-Recomendación por preferencia: Se seleccionan los 5 desafíos con mejor coincidencia. Se implementó de la siguiente forma: sumando los valores absolutos de las diferencias, para cada característica, entre el valor asignado al desafío y la preferencia del usuario en esa característica. Cuanto más baja sea esa suma mayor es el nivel de coincidencia.
-Recomendación por favoritos: Se seleccionan los 20 desafíos con mejor coincidencia, y se los ordena por similitud con el desafío favorito del usuario. Una vez ordenado por similitud, se recomiendan los primeros 5.

Las búsquedas de proyectos se realizan a través de varios filtros individuales:
-Fitro AND
-Filtro OR
-Filtro NOT
-Filtro que incluye una categoría
-Filtro que no incluye una categoría

Elegimos el patrón de diseño composite, debido a que teníamos que cumplir varias condiciones acerca de un proyecto, para que no se vuelva tan tedioso. Nos pareció que con el Patrón Composite se solucionan todos los conflictos posibles. El filtrado utiliza ciertas condiciones iniciales y se indican además los tipos de filtros.
Estos filtros pueden ser combinables mediante los filtros binarios And, Or, Not, los cuales se extienden de la clase abstracta Filter.

# Organización:

En nuestro equipo desglosamos las tareas y utilizamos un kanban para dividirnos las tareas. Por otro lado, los tests doubles, con la herramienta mockito, nos costó implementarlo, así que antes que usar los tests double usamos los objetos de reales como primera solución. Además siempre nos juntábamos los tres mediante Google Meet para poder codear juntos.

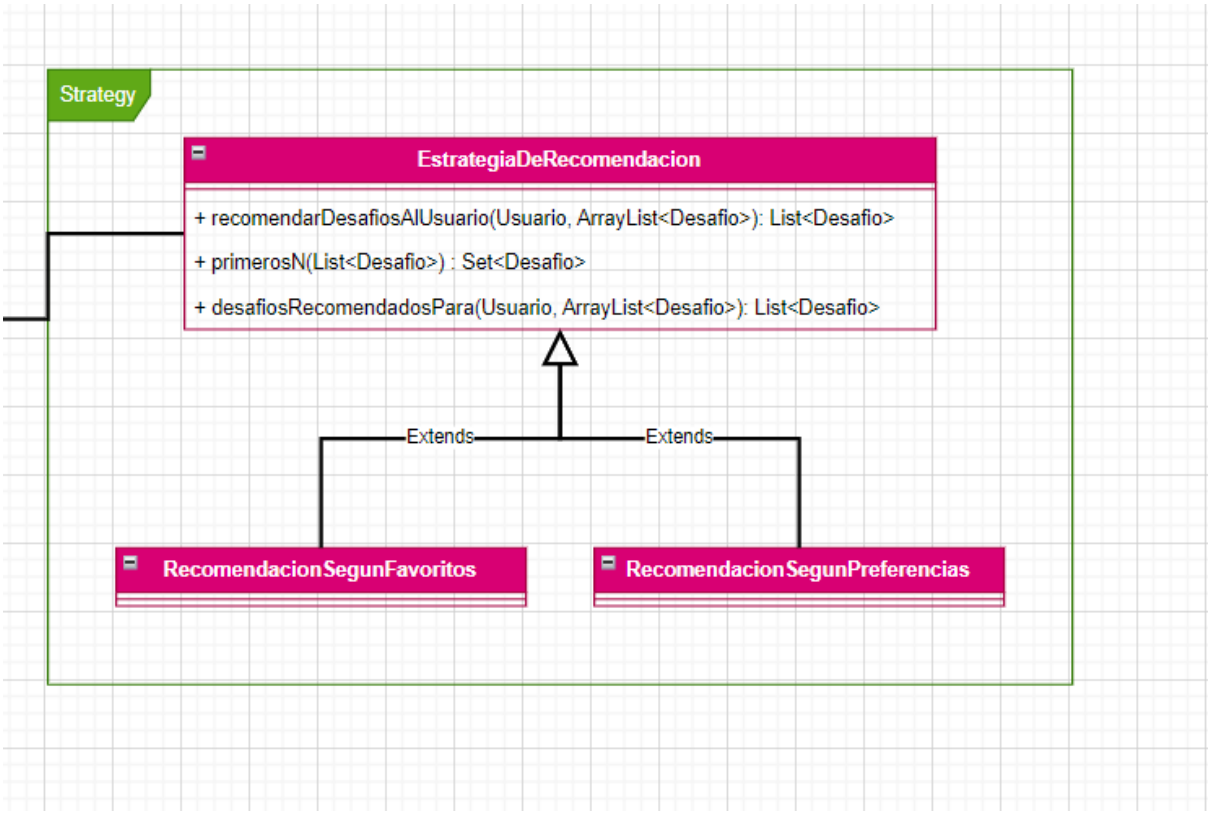
# Patrones implementados

## Patrón Strategy:

**Roles**

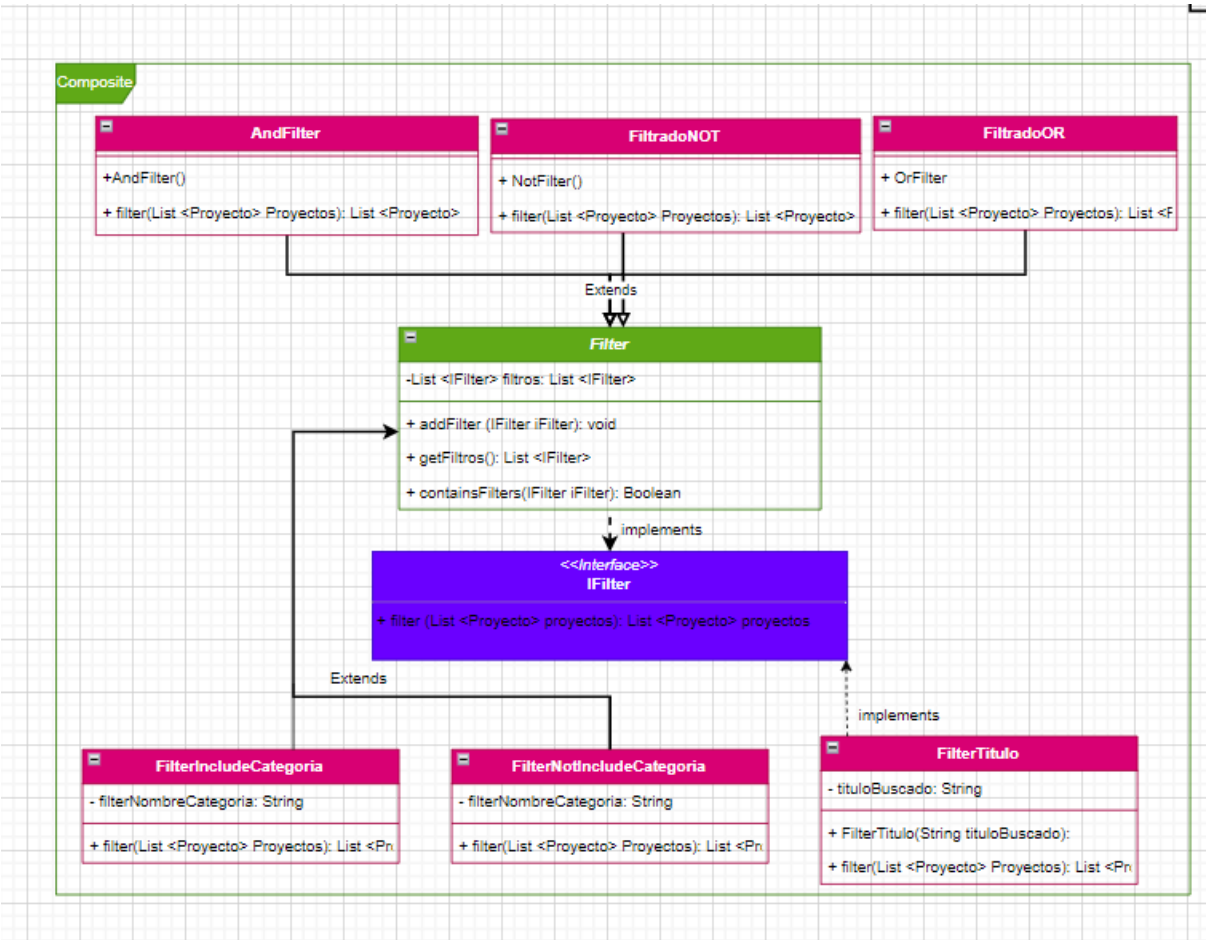
Contexto:Estrategia de recomendación

Estrategia Concreta: Recomendación Según Favoritos, Recomendación Según Preferencias



# Patron Composite:

Roles: -Component -> IFilter
-Composite -> Filter
-CompositeSubClass -> AndFilter, OrFIlter, NotFilter, FilterIncludeCategoria, FilterNotIncludeCategoria
Leaf -> Filter Título



# Patrón State:

**Roles:**

Context -> Usuario
State -> EstadoDelProgreso
State -> ProgresoDeDesafio
Concrete State Subclass -> ProgresoDeDesafioEnCurso, ProgresoDeDesafioTerminado,
ProgresoDeDesafioExpirado